



HAL
open science

Global Constraint Catalog

Nicolas Beldiceanu, Mats Carlsson, Jean-Xavier Rampon

► **To cite this version:**

Nicolas Beldiceanu, Mats Carlsson, Jean-Xavier Rampon. Global Constraint Catalog. 2005. hal-00485396

HAL Id: hal-00485396

<https://hal.science/hal-00485396v1>

Submitted on 20 May 2010

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Global Constraint Catalog

Nicolas Beldiceanu¹
École des Mines de Nantes
LINA, 4 rue Alfred Kastler
BP-20722, FR-44307 Nantes Cedex 3, France

Mats Carlsson
SICS, Box 1263, SE-16 429 Kista, Sweden

Jean-Xavier Rampon
LINA, 2 rue de la Houssinière
BP-92208, FR-44322 Nantes Cedex 3, France

SICS Technical Report T2005:08

ISSN: 1100-3154

ISRN: SICS-T-2005/08-SE

Abstract: This report presents a catalog of global constraints where each constraint is explicitly described in terms of graph properties and/or automata. When available, it also presents some typical usage as well as some pointers to existing filtering algorithms.

Keywords: global constraint, catalog, graph, meta-data.

May 13, 2005

¹Corresponding author, Email: Nicolas.Beldiceanu@emn.fr

Contents

Preface	i
1 Describing global constraints	1
1.1 Describing the arguments of a global constraint	3
1.1.1 Basic data types	3
1.1.2 Compound data types	4
1.1.3 Restrictions	5
1.1.4 Declaring a global constraint	13
1.2 Describing global constraints in terms of graph properties	14
1.2.1 Basic ideas and illustrative example	14
1.2.2 Ingredients used for describing global constraints	16
1.2.3 Graph constraint	42
1.3 Describing global constraints in terms of automata	51
1.3.1 Selecting an appropriate description	51
1.3.2 Defining an automaton	55
2 Description of the catalog	57
2.1 Which global constraints are included?	57
2.2 Which global constraints are missing?	58
2.3 Searching in the catalog	58
2.3.1 How to see if a global constraint is in the catalog?	58
2.3.2 How to search for all global constraints sharing the same structure	59
2.3.3 Searching all places where a global constraint is referenced	60
2.4 Figures of the catalog	61
2.5 Keywords attached to the global constraints	62
3 Further topics	111
3.1 Differences from the 2000 report	111
3.2 Graph invariants	114
3.2.1 Graph classes	115
3.2.2 Format of an invariant	116
3.2.3 Using the database of invariants	117
3.2.4 The database of graph invariants	118
3.3 The electronic version of the catalog	160

4	Global constraint catalog	165
4.1	all_differ_from_at_least_k_pos	172
4.2	all_min_dist	174
4.3	alldifferent	176
4.4	alldifferent_between_sets	180
4.5	alldifferent_except_0	182
4.6	alldifferent_interval	186
4.7	alldifferent_modulo	190
4.8	alldifferent_on_intersection	194
4.9	alldifferent_partition	198
4.10	alldifferent_same_value	200
4.11	allperm	204
4.12	among	206
4.13	among_diff_0	208
4.14	among_interval	212
4.15	among_low_up	214
4.16	among_modulo	218
4.17	among_seq	222
4.18	arith	224
4.19	arith_or	228
4.20	arith_sliding	232
4.21	assign_and_counts	234
4.22	assign_and_nvalues	238
4.23	atleast	242
4.24	atmost	246
4.25	balance	250
4.26	balance_interval	252
4.27	balance_modulo	256
4.28	balance_partition	260
4.29	bin_packing	264
4.30	binary_tree	268
4.31	cardinality_atleast	272
4.32	cardinality_atmost	276
4.33	cardinality_atmost_partition	280
4.34	change	284
4.35	change_continuity	288
4.36	change_pair	298
4.37	change_partition	302
4.38	circuit	306
4.39	circuit_cluster	310
4.40	circular_change	314
4.41	clique	318
4.42	colored_matrix	322
4.43	coloured_cumulative	324
4.44	coloured_cumulatives	328
4.45	common	332

4.46	common_interval	336
4.47	common_modulo	338
4.48	common_partition	340
4.49	connect_points	342
4.50	correspondence	346
4.51	count	350
4.52	counts	354
4.53	crossing	358
4.54	cumulative	362
4.55	cumulative_product	366
4.56	cumulative_two_d	370
4.57	cumulative_with_level_of_priority	374
4.58	cumulatives	378
4.59	cutset	382
4.60	cycle	386
4.61	cycle_card_on_path	390
4.62	cycle_or_accessibility	394
4.63	cycle_resource	398
4.64	cyclic_change	402
4.65	cyclic_change_joker	406
4.66	decreasing	410
4.67	deepest_valley	414
4.68	derangement	418
4.69	differ_from_at_least_k_pos	422
4.70	diffn	426
4.71	diffn_column	430
4.72	diffn_include	432
4.73	discrepancy	434
4.74	disjoint	436
4.75	disjoint_tasks	440
4.76	disjunctive	444
4.77	distance_between	446
4.78	distance_change	448
4.79	domain_constraint	452
4.80	elem	456
4.81	element	460
4.82	element_greatereq	464
4.83	element_lesseq	468
4.84	element_matrix	472
4.85	element_sparse	476
4.86	elements	480
4.87	elements_alldifferent	482
4.88	elements_sparse	486
4.89	eq_set	490
4.90	exactly	492
4.91	global_cardinality	496

4.92	global_cardinality_low_up	500
4.93	global_cardinality_with_costs	502
4.94	global_contiguity	506
4.95	golomb	508
4.96	graph_crossing	512
4.97	group	516
4.98	group_skip_isolated_item	524
4.99	heighest_peak	532
4.100	in	536
4.101	in_relation	538
4.102	in_same_partition	542
4.103	in_set	546
4.104	increasing	548
4.105	indexed_sum	552
4.106	inflexion	554
4.107	int_value_precede	556
4.108	int_value_precede_chain	558
4.109	interval_and_count	560
4.110	interval_and_sum	564
4.111	inverse	568
4.112	inverse_set	572
4.113	ith_pos_different_from_0	576
4.114	k_cut	578
4.115	lex2	580
4.116	lex_alldifferent	582
4.117	lex_between	584
4.118	lex_chain_less	588
4.119	lex_chain_lesseq	592
4.120	lex_different	596
4.121	lex_greater	598
4.122	lex_greatereq	602
4.123	lex_less	606
4.124	lex_lesseq	610
4.125	link_set_to_booleans	614
4.126	longest_change	618
4.127	map	622
4.128	max_index	624
4.129	max_n	626
4.130	max_nvalue	628
4.131	max_size_set_of_consecutive_var	632
4.132	maximum	634
4.133	maximum_modulo	638
4.134	min_index	640
4.135	min_n	644
4.136	min_nvalue	646
4.137	min_size_set_of_consecutive_var	650

4.138	minimum	652
4.139	minimum_except_0	656
4.140	minimum_greater_than	660
4.141	minimum_modulo	664
4.142	minimum_weight_alldifferent	666
4.143	nclass	670
4.144	nequivalence	674
4.145	next_element	676
4.146	next_greater_element	680
4.147	ninterval	682
4.148	no_peak	684
4.149	no_valley	686
4.150	not_all_equal	688
4.151	not_in	690
4.152	npair	694
4.153	nset_of_consecutive_values	696
4.154	nvalue	698
4.155	nvalue_on_intersection	702
4.156	nvalues	704
4.157	nvalues_except_0	706
4.158	one_tree	708
4.159	orchard	712
4.160	orth_link_ori_siz_end	716
4.161	orth_on_the_ground	718
4.162	orth_on_top_of_orth	720
4.163	orths_are_connected	722
4.164	path_from_to	726
4.165	pattern	730
4.166	peak	732
4.167	period	736
4.168	period_except_0	738
4.169	place_in_pyramid	740
4.170	polyomino	744
4.171	product_ctr	748
4.172	range_ctr	750
4.173	relaxed_sliding_sum	752
4.174	same	754
4.175	same_and_global_cardinality	760
4.176	same_intersection	764
4.177	same_interval	766
4.178	same_modulo	768
4.179	same_partition	770
4.180	sequence_folding	772
4.181	set_value_precede	776
4.182	shift	778
4.183	size_maximal_sequence_alldifferent	782

4.184	size_maximal_starting_sequence_alldifferent	784
4.185	sliding_card_skip0	786
4.186	sliding_distribution	790
4.187	sliding_sum	792
4.188	sliding_time_window	794
4.189	sliding_time_window_from_start	798
4.190	sliding_time_window_sum	802
4.191	smooth	806
4.192	soft_alldifferent_ctr	810
4.193	soft_alldifferent_var	814
4.194	soft_same_interval_var	818
4.195	soft_same_modulo_var	820
4.196	soft_same_partition_var	822
4.197	soft_same_var	824
4.198	soft_used_by_interval_var	828
4.199	soft_used_by_modulo_var	832
4.200	soft_used_by_partition_var	836
4.201	soft_used_by_var	838
4.202	sort	842
4.203	sort_permutation	846
4.204	stage_element	850
4.205	stretch_circuit	854
4.206	stretch_path	858
4.207	strict_lex2	862
4.208	strictly_decreasing	864
4.209	strictly_increasing	866
4.210	strongly_connected	868
4.211	sum	870
4.212	sum_ctr	874
4.213	sum_of_weights_of_distinct_values	876
4.214	sum_set	880
4.215	symmetric_alldifferent	882
4.216	symmetric_cardinality	886
4.217	symmetric_gcc	890
4.218	temporal_path	892
4.219	tour	896
4.220	track	900
4.221	tree	902
4.222	tree_range	906
4.223	tree_resource	910
4.224	two_layer_edge_crossing	914
4.225	two_orth_are_in_contact	918
4.226	two_orth_column	922
4.227	two_orth_do_not_overlap	924
4.228	two_orth_include	928
4.229	used_by	930

4.230	used_by_interval	934
4.231	used_by_modulo	936
4.232	used_by_partition	938
4.233	valley	940
4.234	vec_eq_tuple	944
4.235	weighted_partial_alldiff	946

A Legend for the description 949

B Electronic constraint catalog 951

B.1	all_differ_from_at_least_k_pos	957
B.2	all_min_dist	958
B.3	alldifferent	959
B.4	alldifferent_between_sets	960
B.5	alldifferent_except_0	961
B.6	alldifferent_interval	962
B.7	alldifferent_modulo	963
B.8	alldifferent_on_intersection	964
B.9	alldifferent_partition	965
B.10	alldifferent_same_value	967
B.11	allperm	968
B.12	among	969
B.13	among_diff_0	971
B.14	among_interval	973
B.15	among_low_up	975
B.16	among_modulo	977
B.17	among_seq	979
B.18	arith	981
B.19	arith_or	983
B.20	arith_sliding	985
B.21	assign_and_counts	989
B.22	assign_and_nvalues	991
B.23	atleast	993
B.24	atmost	995
B.25	balance	996
B.26	balance_interval	997
B.27	balance_modulo	998
B.28	balance_partition	999
B.29	bin_packing	1000
B.30	binary_tree	1001
B.31	cardinality_atleast	1002
B.32	cardinality_atmost	1003
B.33	cardinality_atmost_partition	1004
B.34	change	1005
B.35	change_continuity	1007
B.36	change_pair	1012

B.37	change_partition	1018
B.38	circuit	1020
B.39	circuit_cluster	1021
B.40	circular_change	1023
B.41	clique	1025
B.42	colored_matrix	1026
B.43	coloured_cumulative	1028
B.44	coloured_cumulatives	1030
B.45	common	1032
B.46	common_interval	1033
B.47	common_modulo	1034
B.48	common_partition	1035
B.49	connect_points	1037
B.50	correspondence	1040
B.51	count	1042
B.52	counts	1044
B.53	crossing	1046
B.54	cumulative	1048
B.55	cumulative_product	1050
B.56	cumulative_two_d	1052
B.57	cumulative_with_level_of_priority	1055
B.58	cumulatives	1057
B.59	cutset	1059
B.60	cycle	1060
B.61	cycle_card_on_path	1061
B.62	cycle_or_accessibility	1063
B.63	cycle_resource	1065
B.64	cyclic_change	1067
B.65	cyclic_change_joker	1069
B.66	decreasing	1072
B.67	deepest_valley	1073
B.68	derangement	1075
B.69	differ_from_at_least_k_pos	1076
B.70	diffn	1078
B.71	diffn_column	1080
B.72	diffn_include	1081
B.73	discrepancy	1082
B.74	disjoint	1083
B.75	disjoint_tasks	1084
B.76	disjunctive	1086
B.77	distance_between	1087
B.78	distance_change	1088
B.79	domain_constraint	1091
B.80	elem	1093
B.81	element	1095
B.82	element_greatereq	1097

B.83	element_lesseq	1099
B.84	element_matrix	1101
B.85	element_sparse	1104
B.86	elements	1106
B.87	elements_alldifferent	1107
B.88	elements_sparse	1109
B.89	eq_set	1111
B.90	exactly	1112
B.91	global_cardinality	1114
B.92	global_cardinality_low_up	1115
B.93	global_cardinality_with_costs	1116
B.94	global_contiguity	1118
B.95	golomb	1120
B.96	graph_crossing	1121
B.97	group	1123
B.98	group_skip_isolated_item	1128
B.99	highest_peak	1132
B.100	in	1134
B.101	in_relation	1136
B.102	in_same_partition	1138
B.103	in_set	1140
B.104	increasing	1141
B.105	indexed_sum	1142
B.106	inflexion	1143
B.107	int_value_precede	1145
B.108	int_value_precede_chain	1146
B.109	interval_and_count	1147
B.110	interval_and_sum	1149
B.111	inverse	1150
B.112	inverse_set	1151
B.113	ith_pos_different_from_0	1153
B.114	k_cut	1155
B.115	lex2	1156
B.116	lex_alldifferent	1157
B.117	lex_between	1158
B.118	lex_chain_less	1161
B.119	lex_chain_lesseq	1162
B.120	lex_different	1163
B.121	lex_greater	1165
B.122	lex_greatereq	1167
B.123	lex_less	1169
B.124	lex_lesseq	1171
B.125	link_set_to_booleans	1173
B.126	longest_change	1174
B.127	map	1176
B.128	max_index	1177

B.129	max_n	1179
B.130	max_nvalue	1180
B.131	max_size_set_of_consecutive_var	1181
B.132	maximum	1182
B.133	maximum_modulo	1184
B.134	min_index	1185
B.135	min_n	1187
B.136	min_nvalue	1188
B.137	min_size_set_of_consecutive_var	1189
B.138	minimum	1190
B.139	minimum_except_0	1192
B.140	minimum_greater_than	1194
B.141	minimum_modulo	1196
B.142	minimum_weight_alldifferent	1197
B.143	nclass	1199
B.144	nequivalence	1200
B.145	next_element	1201
B.146	next_greater_element	1204
B.147	interval	1205
B.148	no_peak	1206
B.149	no_valley	1208
B.150	not_all_equal	1210
B.151	not_in	1212
B.152	npair	1214
B.153	nset_of_consecutive_values	1215
B.154	nvalue	1216
B.155	nvalue_on_intersection	1217
B.156	nvalues	1218
B.157	nvalues_except_0	1219
B.158	one_tree	1220
B.159	orchard	1222
B.160	orth_link_ori_siz_end	1223
B.161	orth_on_the_ground	1224
B.162	orth_on_top_of_orth	1225
B.163	orths_are_connected	1227
B.164	path_from_to	1229
B.165	pattern	1231
B.166	peak	1232
B.167	period	1234
B.168	period_except_0	1235
B.169	place_in_pyramid	1236
B.170	polyomino	1238
B.171	product_ctr	1240
B.172	range_ctr	1241
B.173	relaxed_sliding_sum	1242
B.174	same	1244

B.175	same_and_global_cardinality	1245
B.176	same_intersection	1247
B.177	same_interval	1248
B.178	same_modulo	1249
B.179	same_partition	1250
B.180	sequence_folding	1251
B.181	set_value_precede	1253
B.182	shift	1254
B.183	size_maximal_sequence_alldifferent	1256
B.184	size_maximal_starting_sequence_alldifferent	1257
B.185	sliding_card_skip0	1258
B.186	sliding_distribution	1260
B.187	sliding_sum	1262
B.188	sliding_time_window	1263
B.189	sliding_time_window_from_start	1264
B.190	sliding_time_window_sum	1266
B.191	smooth	1268
B.192	soft_alldifferent_ctr	1270
B.193	soft_alldifferent_var	1271
B.194	soft_same_interval_var	1272
B.195	soft_same_modulo_var	1273
B.196	soft_same_partition_var	1274
B.197	soft_same_var	1276
B.198	soft_used_by_interval_var	1277
B.199	soft_used_by_modulo_var	1278
B.200	soft_used_by_partition_var	1279
B.201	soft_used_by_var	1281
B.202	sort	1282
B.203	sort_permutation	1283
B.204	stage_element	1285
B.205	stretch_circuit	1287
B.206	stretch_path	1289
B.207	strict_lex2	1291
B.208	strictly_decreasing	1292
B.209	strictly_increasing	1294
B.210	strongly_connected	1296
B.211	sum	1297
B.212	sum_ctr	1298
B.213	sum_of_weights_of_distinct_values	1299
B.214	sum_set	1300
B.215	symmetric_alldifferent	1301
B.216	symmetric_cardinality	1302
B.217	symmetric_gcc	1304
B.218	temporal_path	1306
B.219	tour	1308
B.220	track	1310

B.221tree	1312
B.222tree_range	1313
B.223tree_resource	1314
B.224two_layer_edge_crossing	1316
B.225two_orth_are_in_contact	1318
B.226two_orth_column	1320
B.227two_orth_do_not_overlap	1322
B.228two_orth_include	1324
B.229used_by	1326
B.230used_by_interval	1327
B.231used_by_modulo	1328
B.232used_by_partition	1329
B.233valley	1330
B.234vec_eq_tuple	1332
B.235weighted_partial_alldiff	1333
Bibliography	1335
Index	1349

Preface

This catalog presents a list of global constraints. It contains about 235 constraints, which are explicitly described in terms of graph properties and/or automata.

This *Global Constraint Catalog* is an expanded version of the list of global constraints presented in [1]. The principle used for describing global constraints has been slightly modified in order to deal with a larger number of global constraints. Since 2003, we try to provide an automaton that recognizes the solutions associated with a global constraint.

Writing a dictionary is a long process, especially in a field where new words are defined every year. In this context, one difficulty has been related to the fact that we want to express explicitly the meaning of global constraints in terms of meta-data. Finding an appropriate description that easily captures the meaning of most global constraints seems to be a tricky task.

Goal of the catalog. This catalog has four main goals. First, it provides an overview of most of the different global constraints that were gradually introduced in the area of constraint programming since the work of Jean-Louis Laurière on ALICE [2]. It also identifies new global constraints for which no existing published work exists. The global constraints are arranged in alphabetic order, and for all of them a description and an example are systematically provided. When available, it also presents some typical usage as well as some pointers to existing filtering algorithms.

Second, the global constraints described in this catalog are not only accessible to humans, who can read the catalog for searching for some information. It is also available to machines, which can read and interpret it. This is why there exists an electronic version of this catalog where one can get, for most global constraints, a complete description in terms of meta-data. In fact, most of this catalog and its figures were automatically generated from this electronic version by a computer program. This description is based on two complementary ways to look at a global constraint. The first one defines a global constraint as searching for a graph with specific properties [3], while the second one characterizes a global constraint in terms of an automaton that only recognizes the solutions associated to that global constraint [4, 5]. The key point of these descriptions is their ability to define explicitly in a concise way the meaning of most global constraints. In addition these descriptions can also be systematically turned into polynomial filtering algorithms.

Third, we hope that this unified description of apparently diverse global constraints will allow for establishing a systematic link between the properties of basic concepts used for describing global constraints and the properties of the global constraints as a whole.

Finally, we also hope that it will attract more people from the algorithmic community into the area of constraints. To a certain extent this has already started at places like CWI in Amsterdam, the Max-Planck für Informatik (Saarbrücken) or the university of Waterloo.

Use of the catalog. The catalog is organized into four chapters:

- Chapter 1 explains how the meaning of global constraints is described in terms of graph-properties or in terms of automata. On the one hand, if one wants to consult the catalog for getting the informal definition of a global constraint, examples of use of that constraint or pointers to filtering algorithms, then one only needs to read the first section of Chapter 1: Describing the arguments of a global constraint, page 3. On the other hand, if one wants to understand those entries describing explicitly the meaning of a constraint then all the material of Chapter 1 is required.
- Chapter 2 describes the content of the catalog as well as different ways for searching through the catalog. This material is essential.
- Chapter 3 covers additional topics such as the differences from the 2000 report [1] on global constraints, the generation of implied constraints that are systematically linked to the graph-based description of a global constraint, and the electronic version of the catalog. The material describing the format of the entries of a global constraint is mandatory for those who want to exploit the electronic version in order to write preprocessors for performing various tasks for a global constraint.
- Finally, Chapter 4 corresponds to the catalog itself, which gives the global constraints in alphabetical order.

Acknowledgments. Nicolas Beldiceanu was the principal investigator and main architect of the constraint catalog, provided the main ideas, and wrote a checker for the constraint descriptions and the figure generation program for the constraint descriptions.

Jean-Xavier Rampon provided the proofs for the graph invariants.

Mats Carlsson contributed to the design of the meta-data format, generated some of the automata, and wrote the program that created the \LaTeX version of this catalog from the constraint descriptions.

The idea of describing explicitly the meaning of global constraints in a declarative way has been inspired by the work on meta-knowledge of Jacques Pitrat.

We are grateful to Magnus Ågren, Abderrahmane Aggoun, Ernst Althaus, Gregor Baues, Christian Bessière, Éric Bourreau, Pascal Brisset, Hadrien Cambazard, Peter Chan, Philippe Charlier, Evelyne Contejean, Romuald Debruyne, Frédéric Deces, Mehmet Dincbas, François Fage, Pierre Flener, Xavier Gandibleux, Yan Georget, David Hanak, Narendra Jussien, Irit Katriel, Waldemar Kocjan, Per Kreuger, Krzysztof Kuchcinski, Per Mildner, Michel Leconte, Michael Marte, Nicolas Museux, Justin Pearson, Thierry Petit, Emmanuel Poder, Guillaume Rochart, Xavier Savalle, Helmut Simonis, Péter Szeredi, Sven Thiel and Charlotte Truchet for discussion, information exchange or common work about specific global constraints.

Furthermore, we are grateful to Irit Katriel who contributed by updating the description of some filtering algorithms related to flow and matching of the catalog.

Finally, we want to acknowledge the support of SICS and EMN for providing excellent working conditions. The part of this work related to graph properties in Chapter 4 was done while the corresponding author was working at SICS.

Readers may send their suggestion via email to the corresponding author with `catalog` as subject.

Uppsala, Sweden, August 2003

Nantes, France, May 2005

— NB, MC, JXR

Chapter 1

Describing global constraints

Contents

1.1 Describing the arguments of a global constraint	3
1.1.1 Basic data types	3
1.1.2 Compound data types	4
1.1.3 Restrictions	5
1.1.4 Declaring a global constraint	13
1.2 Describing global constraints in terms of graph properties . . .	14
1.2.1 Basic ideas and illustrative example	14
1.2.2 Ingredients used for describing global constraints	16
Collection generators	17
Elementary constraints attached to the arcs	22
Simple arithmetic expressions	22
Arithmetic expressions	23
Arc constraints	24
Graph generators	26
Graph properties	31
Graph terminology and notations	31
Graph characteristics	34
1.2.3 Graph constraint	42
Simple graph constraint	43
Dynamic graph constraint	47
1.3 Describing global constraints in terms of automata	51
1.3.1 Selecting an appropriate description	51
1.3.2 Defining an automaton	55

We first motivate the need for an explicit description of global constraints and then present the *graph-based* as well as the *automaton-based* descriptions used throughout the catalog. On the one hand, the graph-based representation considers a global constraint as a subgraph of an initial given graph. This subgraph has to satisfy a set of

required graph properties. On the other hand, the automaton-based representation denotes a global constraint as a hypergraph constructed from a given constraint checker¹. Both, the initial graph of the graph-based representation, as well as the hypergraph of the automaton-based representation have a very regular structure, which should give the opportunity for efficient filtering algorithms taking advantage of this structure.

We now present our motivations for an explicit description of the meaning of global constraints. The current trend² is to first use natural language for describing the meaning of a global constraint and second to work out a specialized filtering algorithm. Since we have a huge number of potential global constraints that can be combined in a lot of ways, this is an immense task. Since we are also interested in providing other services such as visualization [6], explanations [7], cuts for linear programming [8], moves for local search [9], soft global constraints [10, 11, 12], specialized heuristics for each global constraint this is even worse. One could argue that a candidate for describing explicitly the meaning of global constraints would be second order predicates calculus. This could perhaps solve our description problem but would, at least currently, not be useful for deriving any filtering algorithm. For a similar reason Prolog was restricted to Horn clauses for which one had a reasonable solving mechanism. What we want to stress through this example is the fact that a declarative description is really useful only if it also provides some hints about how to deal with that description. Our first choice of a graph-based representation has been influenced by the following observations:

- The concept of graph takes its roots in the area of mathematical recreations (see for instance L. Euler [13], H. E. Dudeney [14], E. Lucas [15] and T. P. Kirkman [16]), which was somehow the ancestor of combinatorial problems. In this perspective a graph-based description makes sense.
- In one of the first book introducing graph theory [17], C. Berge presents graph theory as a way of grouping apparently diverse problems and results. This was also the case for global constraints.
- The characteristics associated with graphs are concrete and concise.
- Finally, it is well known that graph theory is an important tool with respect to the development of efficient filtering algorithms [18, 19, 20, 21, 22, 23, 24, 25, 26, 27].

Our second choice of an automaton-based representation has been motivated by the following observation. Writing a constraint checker is usually a straightforward task. The corresponding program can usually be turned into an automaton. Of course an automaton is typically used on a fixed sequence of symbols. But, within the context of filtering algorithms, we have to deal with a sequence of variables. For this purpose we have shown [4] for some automata how to decompose them into a conjunction of smaller constraints. In this context, a global constraint can be seen as a hypergraph corresponding to its decomposition.

¹A *constraint checker* is a program that takes an instance of a constraint for which all variables are fixed and tests whether the constraint is satisfied or not.

²This can be observed in all constraint manuals where the description of the meaning is always informal.

1.1 Describing the arguments of a global constraint

Since global constraints have to receive their arguments in some form, no matter whether we use the graph-based or the automaton-based description, we start by describing the abstract data types that we use in order to specify the arguments of a global constraint. These abstract data types are not related to any specific programming language like Caml, C, C++, Java or Prolog. If one wants to focus on a specific language, then one has to map these abstract data types to the data types that are available within the considered programming language. In a second phase we describe all the restrictions that one can impose on the arguments of a global constraint. Finally, in a third phase we show how to use these ingredients in order to declare the arguments of a global constraint.

1.1.1 Basic data types

We provide the following *basic data types*:

- `atom` corresponds to an atom. Predefined atoms are `MININT` and `MAXINT`, which respectively correspond to the *smallest* and to the *largest integer*.
- `int` corresponds to an *integer value*.
- `dvar` corresponds to a *domain variable*. A *domain variable* is a variable that will be assigned an *integer* value taken from an initial finite set of integer values.
- `sint` corresponds to a *finite set of integer values*.
- `svar` corresponds to a *set variable*. A *set variable* is a variable that will be assigned to a *finite set* of integer values.
- `mint` corresponds to a *multiset of integer values*.
- `mvar` corresponds to a *multiset variable*. A *multiset variable* is a variable that will be assigned to a *multiset of integer values*.
- `flt` corresponds to a *float number*.
- `fvar` corresponds to a *float variable*. A *float variable* is a variable that will be assigned a *float number* taken from an initial finite set of intervals.

1.1.2 Compound data types

We provide the following *compound data types*:

- $\text{list}(T)$ corresponds to a list of elements of type T , where T is a basic or a compound data type.
- $c : \text{collection}(A_1, A_2, \dots, A_n)$ corresponds to a collection c of ordered items, where each item consists of $n > 0$ attributes A_1, A_2, \dots, A_n . Each attribute is an expression of the form $a - T$, where a is the *name* of the attribute and T the *type* of the attribute (a basic or a compound data type). All names of the attributes of a given collection should be distinct and different from the keyword *key*, which corresponds to an implicit³ attribute. Its value corresponds to the position of an item within the collection. The first item of a collection is associated with position 1.

The following notations are used for instantiated arguments:

- A list of elements e_1, e_2, \dots, e_n is denoted $[e_1, e_2, \dots, e_n]$.
- A finite set of integers i_1, i_2, \dots, i_n is denoted $\{i_1, i_2, \dots, i_n\}$.
- A multiset of integers i_1, i_2, \dots, i_n is denoted $\{\{i_1, i_2, \dots, i_n\}\}$.
- A collection of n items, each item having m attributes, is denoted by $\{\mathbf{a}_1 - v_{11} \dots \mathbf{a}_m - v_{1m}, \mathbf{a}_1 - v_{21} \dots \mathbf{a}_m - v_{2m}, \dots, \mathbf{a}_1 - v_{n1} \dots \mathbf{a}_m - v_{nm}\}$. Each item is separated from the previous item by a comma.
- The i^{th} item of a collection c is denoted $c[i]$.
- The number of items of a collection c is denoted $|c|$.

³This attribute is not explicitly defined.

EXAMPLE: Let us illustrate with three examples, the types one can create. These examples concern the creation of a collection of variables, a collection of tasks and a collection of orthotopes^a.

- In the first example we define VARIABLES so that it corresponds to a collection of variables. VARIABLES is for instance used in the `alldifferent` constraint. The declaration `VARIABLES : collection(var – dvar)` defines a collection of items, each of which having one attribute `var` that is a domain variable.
- In the second example we define TASKS so that it corresponds to a collection of tasks, each task being defined by its origin, its duration, its end and its resource consumption. Such a collection is for instance used in the `cumulative` constraint. The declaration `TASKS : collection(origin – dvar, duration – dvar, end – dvar, height – dvar)` defines a collection of items, each of which having the four attributes `origin`, `duration`, `end` and `height` which all are domain variables.
- In the last example we define ORTHOTOPEs so that it corresponds to a collection of orthotopes. Each orthotope is described by an attribute `orth`. Unlike the previous examples, the type of this attribute does not correspond any more to a basic data type but rather to a collection of n items, where n is the number of dimensions of the orthotope^b. This collection, named ORTHOTOPE, defines for a given dimension the origin, the size and the end of the object in this dimension. This leads to the two declarations:

```
– ORTHOTOPE – collection(ori – dvar, siz – dvar, end – dvar),
– ORTHOTOPEs – collection(orth – ORTHOTOPE).
```

ORTHOTOPE is for instance used in the `diffn` constraint.

^aAn *orthotope* corresponds to the generalization of a segment, a rectangle and a box to the n -dimensional case.

^b1 for a segment, 2 for a rectangle, 3 for a box,

1.1.3 Restrictions

When defining the arguments of a global constraint, it is often the case that one needs to express additional conditions that refine the type declaration of its arguments. For this purpose we provide *restrictions* that allow for specifying these additional conditions. Each restriction has a name and a set of arguments and is described by the following items:

- A small paragraph first describes the effect of the restriction,
- An example points to a constraint using the restriction,
- Finally, a ground instance, preceded by the symbol \triangleright , which satisfies the restriction is given. Similarly, a ground instance, preceded by the symbol \blacktriangleright , which violates the restriction is proposed. In this latter case, a bold font may be used for pointing to the source of the problem.

Currently the list of restrictions is:

- `in_list(Arg, ListAtoms)`:
 - `Arg` is an argument of type `atom`,
 - `ListAtoms` is a non-empty list of distinct atoms.

This restriction forces `Arg` to be one of the atoms specified in the list `ListAtoms`.

EXAMPLE: An example of use of such restriction can be found in the `change(NCHANGE, VARIABLES, CTR)` constraint: `in_list(CTR, [=, ≠, <, ≥, >, ≤])` forces the last argument `CTR` of the `change` constraint to take its value in the list of atoms `[=, ≠, <, ≥, >, ≤]`.

▷ `change(1, {var - 4, var - 4, var - 4, var - 6}, ≠)`
 ► `change(1, {var - 4, var - 4, var - 4, var - 6}, 3)`

- `in_list(Arg, Attr, ListInt)`:
 - `Arg` is an argument of type `collection`,
 - `Attr` is an attribute of type `int` of the collection denoted by `Arg`,
 - `ListInt` is a non-empty list of integers.

This restriction enforces for all items of the collection `Arg`, the attribute `Attr` to take its value within the list of integers `ListInt`.

EXAMPLE: An example of use of such restriction can be found in the `one_tree` constraint: `in_list(NODES, type, [2, 3, 6])` forces the attribute `type` of the `NODES` collection to take its value in the list of integers `[2, 3, 6]`.

▷ `one_tree({ id - a index - 1 type - 2 father - 1 depth1 - 1 depth2 - 0, id - b index - 2 type - 2 father - 2 depth1 - 0 depth2 - 0, id - c index - 3 type - 3 father - 2 depth1 - 0 depth2 - 0, id - d index - 4 type - 3 father - 2 depth1 - 0 depth2 - 0})`
 ► `one_tree({ id - a index - 1 type - 9 father - 1 depth1 - 1 depth2 - 0, id - b index - 2 type - 2 father - 2 depth1 - 0 depth2 - 0, id - c index - 3 type - 3 father - 2 depth1 - 0 depth2 - 0, id - d index - 4 type - 3 father - 2 depth1 - 0 depth2 - 0})`

- `in_attr(Arg1, Attr1, Arg2, Attr2)`:
 - `Arg1` is an argument of type `collection`,
 - `Attr1` is an attribute of type `dvar` of the collection denoted by `Arg1`,
 - `Arg2` is an argument of type `collection`,
 - `Attr2` is an attribute of type `int` of the collection denoted by `Arg2`.

Let \mathcal{S}_2 denote the set of values assigned to the `Attr2` attributes of the items of the collection `Arg2`. This restriction enforces the following condition: For all items of the collection `Arg1`, the attribute `Attr1` takes its value in the set \mathcal{S}_2 .

EXAMPLE: An example of use of such restriction can be found in the `cumulatives(TASKS, MACHINES, CTR)` constraint: `in_attr(TASKS, machine, MACHINES, id)` enforces that the `machine` attribute of each task of the `TASKS` collection correspond to a machine identifier (i.e. an `id` attribute of the `MACHINES` collection).

```
▷cumulatives({ machine - 1 origin - 2 duration - 2 end - 4 height - 2,
              machine - 1 origin - 2 duration - 2 end - 4 height - 2,
              machine - 2 origin - 1 duration - 4 end - 5 height - 5,
              machine - 1 origin - 4 duration - 2 end - 6 height - 1},
            {id - 1 capacity - 9, id - 2 capacity - 8}, ≤)
▶cumulatives({ machine - 5 origin - 2 duration - 2 end - 4 height - 2,
              machine - 1 origin - 2 duration - 2 end - 4 height - 2,
              machine - 2 origin - 1 duration - 4 end - 5 height - 5,
              machine - 1 origin - 4 duration - 2 end - 6 height - 1},
            {id - 1 capacity - 9, id - 2 capacity - 8}, ≤)
```

- `distinct(Arg, Attrs)`:
 - `Arg` is an argument of type collection,
 - `Attrs` is an attribute of type `int` or a list of distinct attributes of type `int` of the collection denoted by `Arg`.

For all pairs of distinct items of the collection `Arg` this restriction enforces that there be at least one attribute specified by `Attrs` with two distinct values.

EXAMPLE: An example of use of such restriction can be found in the `cycle(NCYCLE, NODES)` constraint: `distinct(NODES, index)` enforces that all `index` attributes of the `NODES` collection take distinct values.

```
▷cycle(2, {index - 1 succ - 2, index - 2 succ - 1, index - 3 succ - 3})
▶cycle(2, {index - 1 succ - 2, index - 1 succ - 1, index - 3 succ - 3})
```

- `increasing_seq(Arg, Attrs)`:
 - `Arg` is an argument of type collection,
 - `Attrs` is an attribute of type `int` or a list of distinct attributes of type `int` of the collection denoted by `Arg`.

Let n and m respectively denote the number of items of the collection `Arg`, and the number of attributes of `Attrs`. For the i^{th} item of the collection `Arg` let t_i denote the tuple of values $\langle v_{i,1}, v_{i,2}, \dots, v_{i,m} \rangle$ where $v_{i,j}$ is the value of the j^{th} attribute of `Attrs` of the i^{th} item of `Arg`. The restriction enforces a strict lexicographical ordering on the tuples t_1, t_2, \dots, t_n .

EXAMPLE: An example of use of such restriction can be found in the `element_matrix(MAX_I, MAX_J, INDEX_I, INDEX_J, MATRIX, VALUE)` constraint: `increasing_seq(MATRIX, [i, j])` enforces that all items of the `MATRIX` collection be sorted in strictly increasing lexicographic order on the pair (i, j) .

```

▷ element_matrix(2, 2, 1, 2, {i - 1 j - 1 v - 4, i - 1 j - 2 v - 7,
                             i - 2 j - 1 v - 1, i - 2 j - 2 v - 1}, 7)
▶ element_matrix(2, 2, 1, 2, {i - 1 j - 2 v - 4, i - 1 j - 1 v - 7,
                             i - 2 j - 1 v - 1, i - 2 j - 2 v - 1}, 7)

```

- `required(Arg, Attrs)`:
 - `Arg` is an argument of type collection,
 - `Attrs` is an attribute or a list of distinct attributes of the collection denoted by `Arg`.

This restriction enforces that all attributes denoted by `Attrs` be explicitly used within all items of the collection `Arg`.

EXAMPLE: An example of use of such restriction can be found in the `cumulative(TASKS, LIMIT)` constraint: `required(TASKS, height)` enforces that all items of the `TASKS` collection mention the `height` attribute.

```

▷ cumulative({ origin - 2 duration - 2 end - 4 height - 2,
              origin - 2 duration - 2 end - 4 height - 2,
              origin - 1 duration - 4 end - 5 height - 5,
              origin - 4 duration - 2 end - 6 height - 1}, 12)
▶ cumulative({ origin - 2 duration - 2 end - 4,
              origin - 2 duration - 2 end - 4 height - 2,
              origin - 1 duration - 4 end - 5 height - 5,
              origin - 4 duration - 2 end - 6 height - 1}, 12)

```

The `required` restriction is usually systematically used for every attribute of a collection. It is not used when some attributes may be implicitly defined according to other attributes. In this context, we use the `require_at_least` restriction, which we now introduce.

- `require_at_least(Atleast, Arg, Attrs)`:
 - `Atleast` is a positive integer,
 - `Arg` is an argument of type collection,
 - `Attrs` is a non-empty list of distinct attributes of the collection denoted by `Arg`. The length of this list should be strictly greater than `Atleast`.

This restriction enforces that at least `Atleast` attributes of the list `Attrs` be explicitly used within all items of the collection `Arg`.

EXAMPLE: An example of use of such restriction can be found in the `cumulative(TASKS, LIMIT)` constraint:
`require_at_least(2, TASKS, [origin, duration, end])` enforces that all items of the `TASKS` collection mention at least two attributes from the list of attributes `[origin, duration, end]`. In this context, this stems from the fact that we have the equality `origin + duration = end`. This allows for retrieving the third attribute from the values of the two others.

```
▷cumulative({ origin - 2 duration - 2 height - 2,
             origin - 2 end - 4 height - 2,
             duration - 4 end - 5 height - 5,
             origin - 4 duration - 2 end - 6 height - 1}, 12)
▶cumulative({ origin - 2 height - 2,
             origin - 2 duration - 2 end - 4 height - 2,
             origin - 1 duration - 4 end - 5 height - 5,
             origin - 4 duration - 2 end - 6 height - 1}, 12)
```

- `same_size(Arg, Attr)`:
 - `Arg` is an argument of type collection,
 - `Attr` is an attribute of the collection denoted by `Arg`. This attribute should be of type collection.

This restriction enforces that all collections denoted by `Attr` have the same number of items.

EXAMPLE: An example of use of such restriction can be found in the `diffn(ORTHOTOPES)` constraint^a: `same_size(ORTHOTOPES, orth)` forces all the items of the `ORTHOTOPES` collection to be constituted from the same number of items (of type `ORTHOTOPE`). From a practical point of view, this forces the `diffn` constraint to take as its argument a set of points, a set of rectangles, a set of parallelepipeds, ...

```
▷diffn({ {orth - {ori - 2 siz - 2 end - 4, ori - 1 siz - 3 end - 4},
          orth - {ori - 4 siz - 4 end - 8, ori - 3 siz - 3 end - 3},
          orth - {ori - 9 siz - 2 end - 11, ori - 4 siz - 3 end - 7}}
▶diffn({ {orth - {ori - 2 siz - 2 end - 4},
          orth - {ori - 4 siz - 4 end - 8, ori - 3 siz - 3 end - 3},
          orth - {ori - 9 siz - 2 end - 11, ori - 4 siz - 3 end - 7}})
```

^aORTHOTOPES corresponds to the third item of the example presented at page 5.

- `Term1 Comparison Term2`:
 - `Term1` is a *term*. A *term* is an expression that can be evaluated to one or possibly several integer values. The expressions we allow for a *term* are defined in the next paragraph.
 - Comparison is one of the following comparison operators \leq , \geq , $<$, $>$, $=$, \neq .
 - `Term2` is a *term*.

Let $v_{1,1}, v_{1,2}, \dots, v_{1,n_1}$ and $v_{2,1}, v_{2,2}, \dots, v_{2,n_2}$ be the values respectively associated with Term_1 and with Term_2 . The restriction Term_1 Comparison Term_2 forces $v_{1,i}$ Comparison $v_{2,j}$ to hold for every $i \in [1, n_1]$ and every $j \in [1, n_2]$.

A *term* is one of the following expressions:

- e , where e is an integer. The corresponding value is e .
- $|c|$, where c is an argument of type collection. The value of $|c|$ is the number of items of the collection denoted by c .

EXAMPLE: This kind of expression is for instance used in the restrictions of the `atleast(N, VARIABLES, VALUE)` constraint: $N \leq |\text{VARIABLES}|$ restricts N to be less than or equal to the number of items of the `VARIABLES` collection.

```
> atleast(2, {var - 5, var - 8, var - 5}, 5)
▶ atleast(4, {var - 5, var - 8, var - 5}, 5)
```

- `min_size(c, a)`, where c is an argument of type collection and a an attribute of c of type collection. The value of `min_size(c, a)` is the smallest number of items over all collections denoted by a .

EXAMPLE: This kind of expression is for instance used in the restrictions of the `in_relation(VARIABLES, TUPLES_OF_VALS)` constraint: `min_size(TUPLES_OF_VALS, tuple) = |VARIABLES|` forces the smallest number of items associated with the `tuple` attribute to equal the number of items of the `VARIABLES` collection.

```
> in_relation({{var - 5, var - 3, var - 3},
               {tuple - {val - 5, val - 2, val - 3},
                tuple - {val - 5, val - 2, val - 6},
                tuple - {val - 5, val - 3, val - 3}}})
▶ in_relation({{var - 5, var - 3, var - 3},
               {tuple - {val - 5, val - 2},
                tuple - {val - 5, val - 2, val - 6},
                tuple - {val - 5, val - 3, val - 3}}})
```

- `max_size(c, a)`, where c is an argument of type collection and a an attribute of c of type collection. The value of `max_size(c, a)` is the largest number of items over all collections denoted by a .

EXAMPLE: This kind of expression is for instance used in the restrictions of the `in_relation(VARIABLES, TUPLES_OF_VALS)` constraint: `max_size(TUPLES_OF_VALS, tuple) = |VARIABLES|` forces the largest number of items associated with the `tuple` attribute to equal the number of items of the `VARIABLES` collection.

```
> in_relation({{var - 5, var - 3, var - 3},
               {tuple - {val - 5, val - 2, val - 3},
                tuple - {val - 5, val - 2, val - 6},
                tuple - {val - 5, val - 3, val - 3}}})
▶ in_relation({{var - 5, var - 3, var - 3},
               {tuple - {val - 5, val - 2, val - 8, val - 2},
                tuple - {val - 5, val - 2, val - 6},
                tuple - {val - 5, val - 3, val - 3}}})
```

- t , where t is an argument of type `int`. The value of t is the value of the corresponding argument.

EXAMPLE: This kind of expression is for instance used in the restrictions of the `atleast(N, VARIABLES, VALUE)` constraint: $N \geq 0$ forces the first argument of the `atleast` constraint to be greater than or equal to 0.

```
▷ atleast(2, {var - 5, var - 8, var - 5}, 5)
▶ atleast(-1, {var - 5, var - 8, var - 5}, 5)
```

- v , where v is an argument of type `dvar`. The value of v will be the value assigned to variable v ⁴.

EXAMPLE: This kind of expression is for instance used in the restrictions of the `among(NVAR, VARIABLES, VALUES)` constraint: $NVAR \geq 0$ forces the first argument of the `among` constraint to be greater than or equal to 0.

```
▷ among(2, {var - 5, var - 8, var - 5}, {val - 1, val - 5})
▶ among(-9, {var - 5, var - 8, var - 5}, {val - 1, val - 5})
```

- $c.a$, where c is an argument of type `collection` and a an attribute of c of type `int` or `dvar`. The values denoted by $c.a$ are all the values corresponding to attribute a for the different items of c . When $c.a$ designates a domain variable we consider the value assigned to that variable.

EXAMPLE: This kind of expression is for instance used in the restrictions of the `cumulative(TASKS, LIMIT)` constraint: `TASKS.duration` ≥ 0 enforces for all items of the `TASKS` collection that the `duration` attribute be greater than or equal to 0.

```
▷ cumulative({ origin - 2 duration - 2 end - 4 height - 2,
              origin - 2 duration - 2 end - 4 height - 2,
              origin - 1 duration - 4 end - 5 height - 5,
              origin - 4 duration - 2 end - 6 height - 1}, 12)
▶ cumulative({ origin - 2 duration - -2 end - 4 height - 2,
              origin - 2 duration - 2 end - 4 height - 2,
              origin - 1 duration - 4 end - 5 height - 5,
              origin - 4 duration - 2 end - 6 height - 1}, 12)
```

- $c.a$, where c is an argument of type `collection` and a an attribute of c of type `sint` or `svar`. The values denoted by $c.a$ are all the values belonging to the sets corresponding to attribute a for the different items of c . When $c.a$ designates a set variable we consider the values that finally belong to that set.

⁴This stems from the fact that restrictions are defined on the ground instance of a global constraint.

EXAMPLE: This kind of expression is for instance used in the restrictions of the `inverse_set(X, Y)` constraint: $X.x \geq 1$ enforces for all items of the `X` collection that all the potential elements of the set variable associated with the `x` attribute be greater than or equal to 1.

```
▷ inverse_set({ index - 1 x - {2, 4}, index - 2 x - {4},
              index - 3 x - {1},   index - 4 x - {4} },
             { index - 1 y - {3},   index - 2 y - {1},
               index - 3 y - {},     index - 4 y - {1, 2, 4},
               index - 5 y - {} })
► inverse_set({ index - 1 x - {0, 2, 4}, index - 2 x - {4},
              index - 3 x - {1},   index - 4 x - {4} },
             { index - 1 y - {3},   index - 2 y - {1},
               index - 3 y - {},     index - 4 y - {1, 2, 4},
               index - 5 y - {} })
```

- $\min(t_1, t_2)$ or $\max(t_1, t_2)$, where t_1 and t_2 are *terms*. Let \mathcal{V}_1 and \mathcal{V}_2 denote the sets of values respectively associated with the terms t_1 and t_2 . Let $\min(\mathcal{V}_1)$, $\max(\mathcal{V}_1)$ and $\min(\mathcal{V}_2)$, $\max(\mathcal{V}_2)$ denote the minimum and maximum values of \mathcal{V}_1 and \mathcal{V}_2 . The value associated with $\min(t_1, t_2)$ is $\min(\min(\mathcal{V}_1), \min(\mathcal{V}_2))$, while the value associated with $\max(t_1, t_2)$ is $\max(\max(\mathcal{V}_1), \max(\mathcal{V}_2))$.

EXAMPLE: This kind of expression is for instance used in the restrictions of the `ninterval(NVAL, VARIABLES, SIZE_INTERVAL)` constraint: $NVAL \geq \min(1, |\text{VARIABLES}|)$ forces `NVAL` to be greater than or equal to the minimum of 1 and the number of items of the `VARIABLES` collection.

```
▷ ninterval(2, {var - 3, var - 1, var - 9, var - 1, var - 9}, 4)
► ninterval(0, {var - 3, var - 1, var - 9, var - 1, var - 9}, 4)
```

- $t_1 \text{ op } t_2$, where t_1 and t_2 are *terms* and `op` one of the operators `+`, `-`, `*` or `/`⁵. Let \mathcal{V}_1 and \mathcal{V}_2 denote the sets of values respectively associated with the terms t_1 and t_2 . The set of values associated with $t_1 \text{ op } t_2$ is $\mathcal{V}_{12} = \{v : v = v_1 \text{ op } v_2, v_1 \in \mathcal{V}_1, v_2 \in \mathcal{V}_2\}$.

EXAMPLE: This kind of expression is for instance used in the restrictions of the `relaxed_sliding_sum(ATLEAST, ATMOST, LOW, UP, SEQ, VARIABLES)` constraint: $\text{ATMOST} \leq |\text{VARIABLES}| - \text{SEQ} + 1$ forces `ATMOST` to be less than or equal to an arithmetic expression that corresponds to the number of sequences of `SEQ` consecutive variables in a sequence of `|VARIABLES|` variables.

```
▷ relaxed_sliding_sum(3, 4, 3, 7, 4, {var - 2, var - 4, var - 2, var - 0,
                                       var - 0, var - 3, var - 4})
► relaxed_sliding_sum(3, 9, 3, 7, 4, {var - 2, var - 4, var - 2, var - 0,
                                       var - 0, var - 3, var - 4})
```

- Finally, we can also use a constraint C of the catalog for expressing a restriction as long as that constraint is not defined according to the constraint under consideration. The constraint C should have a graph-based or an automaton-based description so that its meaning is explicitly defined.

⁵/ denotes an integer division, a division in which the fractional part is discarded.

EXAMPLE: An example of use of such restriction can be found in the `sort_permutation(FROM, PERMUTATION, TO)` constraint: `alldifferent(PERMUTATION)` is used to express the fact that the variables of the second argument of the `sort_permutation` constraint should take distinct values.

1.1.4 Declaring a global constraint

Declaring a global constraint consists of providing the following information:

- A term $\text{ctr}(A_1, A_2, \dots, A_n)$, where `ctr` corresponds to the *name* of the global constraint and A_1, A_2, \dots, A_n to its *arguments*.
- A possibly empty list of *type declarations*, where each declaration has the form `type:type_declaration`; `type` is the *name* of the new type we define and `type_declaration` is a basic data type, a compound data type or a type previously defined.
- An *argument declaration* $A_1:T_1, A_2:T_2, \dots, A_n:T_n$ giving for each argument A_1, A_2, \dots, A_n of the global constraint `ctr` its type. Each type is a basic data type, a compound data type, or a type that was declared in the list of type declarations.
- A possibly empty *list of restrictions*, where each restriction is one of the restrictions described in Section 1.1.3 (page 5).

EXAMPLE: The arguments of the `all_differ_from_at_least_k_pos` constraint are described by:

Constraint	<code>all_differ_from_at_least_k_pos(K, VECTORS)</code>
Type(s)	<code>VECTOR</code> – <code>collection(var – dvar)</code>
Argument(s)	<code>K</code> – <code>int</code> <code>VECTORS</code> – <code>collection(vec – VECTOR)</code>
Restriction(s)	<code>required(VECTOR, var)</code> <code>K ≥ 0</code> <code>required(VECTORS, vec)</code> <code>same_size(VECTORS, vec)</code>

The first line indicates that the `all_differ_from_at_least_k_pos` constraint has two arguments: `K` and `VECTORS`. The second line declares a new type `VECTOR`, which corresponds to a collection of variables. The third line indicates that the first argument `K` is an integer, while the fourth line tells that the second argument `VECTORS` corresponds to a collection of vectors of type `VECTOR`. Finally the four restrictions respectively enforce that:

- All the items of the `VECTOR` collection mention the `var` attribute,
- `K` be greater than or equal to 0,
- All the items of the `VECTORS` collection mention the `vec` attribute,
- All the vectors have the same number of components.

1.2 Describing global constraints in terms of graph properties

Through a practical example, we first present in a simplified form the basic principles used for describing the meaning of global constraints in terms of graph properties. We then give the full details about the different features used in the description process.

1.2.1 Basic ideas and illustrative example

Within the graph-based representation, a global constraint is represented as a digraph where each vertex corresponds to a variable and each arc to a binary arc constraint between the variables associated with the extremities of the corresponding arc. The main difference with classical constraint networks [28], stems from the fact that we don't force any more all arc constraints to hold. We rather consider this graph from which we discard all the arc constraints that do not hold and impose one or several graph properties on this remaining graph. These properties can for instance be a restriction on the number of connected components, on the size of the smallest connected component or on the size of the largest connected component.

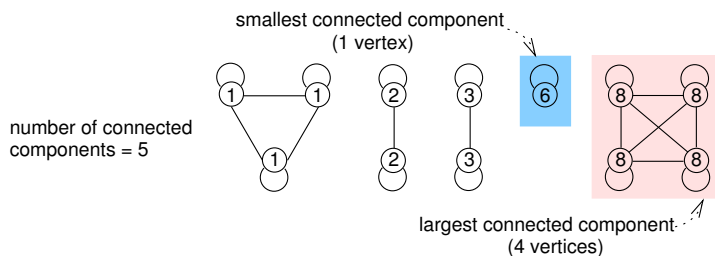


Figure 1.1: Illustration of the link between graph-properties and global constraints

EXAMPLE: We give an example of interpretation of such graph properties in terms of global constraints. For this purpose we consider the sequence s of values 1 3 1 1 2 8 8 2 3 6 8 8 3 from which we construct the following graph G :

- To each value associated with a position in s corresponds a vertex of G ,
- There is an arc from a vertex v_1 to a vertex v_2 if these vertices correspond to the same value.

Figure 1.1 depicts graph G . Since G is symmetric, we omit the directions of the arcs. We have the following correspondence between graph properties and constraints on the sequence s :

- The number of connected components of G corresponds to the number of distinct values of s .
- The size of the smallest connected component of G is the smallest number of occurrences of the same value in s .
- The size of the largest connected component of G is the largest number of occurrences of the same value in s .

As a result, in this context, putting a restriction on the number of connected components of G can be seen as a global constraint on the number of distinct values of a sequence of variables. Similar global constraints can be associated with the two other graph properties.

We now explain how to generate the initial graph associated with a global constraint. A global constraint has one or more arguments, which usually correspond to an integer value, to one variable or to a collection of variables. Therefore we have to describe the process that allows for generating the vertices and the arcs of the initial graph from the arguments of a global constraint under consideration. For this purpose we will take a concrete example.

Consider the constraint `nvalue(NVAL, VARIABLES)` where `NVAL` and `VARIABLES` respectively correspond to a domain variable and to a collection of domain variables $\{\text{var} - V_1, \text{var} - V_2, \dots, \text{var} - V_m\}$ ⁶. This constraint holds if `NVAL` is equal to the number of distinct values assigned to the variables V_1, V_2, \dots, V_m . We first show how to generate the initial graph associated with the `nvalue` constraint. We then describe the arc constraint associated with each arc of this graph. Finally, we give the graph characteristic we impose on the final graph.

To each variable of the collection `VARIABLES` corresponds a vertex of the initial graph. We generate an arc between each pair of vertices. To each arc, we associate an equality constraint between the variables corresponding to the extremities of that arc. We impose that `NVAL`, the variable corresponding to the first argument of `nvalue`, be equal to the number of strongly connected components of the final graph. This final graph consists of the initial graph from which we discard all arcs such that the corresponding equality constraint does not hold.

Part (A) of Figure 1.2 shows the graph initially generated for the constraint `nvalue(NVAL, {var - V1, var - V2, var - V3, var - V4})`, where `NVAL`, V_1 , V_2 , V_3 and V_4 are domain variables. Part (B) presents the final graph associated with the ground instance `nvalue(3, {var - 5, var - 5, var - 1, var - 8})`. For each vertex of the initial and final

⁶`var` corresponds to the name of the attribute used in the collection of variables.

graph we respectively indicate the corresponding variable and the value assigned to that variable. We have removed from the final graph all the arcs associated to equalities that do not hold. The constraint `nvalue(3, {var-5, var-5, var-1, var-8})` holds since the final graph contains three strongly connected components, which, in the context of the definition of the `nvalue` constraint, can be reinterpreted as the fact that `NVAL` is the number of distinct values assigned to variables V_1, V_2, V_3, V_4 .

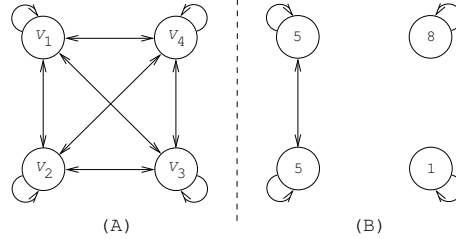


Figure 1.2: Initial and final graph associated with `nvalue`

Now that we have illustrated the basic ideas for describing a global constraint in terms of graph properties, we go into more details.

1.2.2 Ingredients used for describing global constraints

We first introduce the basic ingredients used for describing a global constraint and illustrate them shortly on the example of the `nvalue` constraint introduced in the previous section (page 15). We then go through each basic ingredient in more detail. The graph-based description is founded on the following basic ingredients:

- *Data types and restrictions* used in order to describe the arguments of a global constraint. Data types and restrictions were already described in the previous section (from page 3 to page 13).
- *Collection generators* used in order to derive new collections from the arguments of a global constraint for one of the following reasons:
 - Collection generators are sometimes required since the initial graph of a global constraint cannot always be directly generated from the arguments of the global constraint. The `nvalue(NVAL, VARIABLES)` constraint did not require any collection generator since the vertices of its initial graph were directly generated from the `VARIABLES` collection.
 - A second use of collection generators is for deriving a collection of items for different set of vertices of the final graph. This is sometimes required when we use *set generators* (see the last item of the enumeration).
- *Elementary constraints* associated with the arcs of the initial and final graph of a global constraint. The `nvalue` constraint was using an *equality* constraint, but other constraints are usually required.

1.2. DESCRIBING GLOBAL CONSTRAINTS IN TERMS OF GRAPH PROPERTIES¹⁷

- *Graph generators* employed for constructing the initial graph of a global constraint. In the context of the `nvalue` constraint the initial graph was a *clique*. As we will see later, other patterns are needed for generating an initial graph.
- *Graph characteristics* used for constraining the final graph we want to obtain. In the context of the `nvalue` constraint we were using the *number of strongly connected components* for expressing the fact that we want to count the number of distinct values.
- *Set generators* which may be used for generating specific sets of vertices of the final graph on which we want to enforce a given constraint. Since the `nvalue` constraint enforces a graph property on the final graph (and not on subparts of the final graph) we did not use this feature.

We first start to explain each ingredient separately and then show how one can describe most global constraints in terms of these basic ingredients.

Collection generators

The vertices of the initial graph are usually directly generated from collections of items that are arguments of the global constraint `G` under consideration. However, it sometimes happens that we would like to derive a new collection from existing arguments of `G` in order to produce the vertices of the initial graph.

EXAMPLE: This is for instance the case of the `element(INDEX, TABLE, VALUE)` constraint, where `INDEX` and `VALUE` are domain variables that we would like to group as a single item \mathcal{I} (with two attributes) of a new derived collection. This is in fact done in order to generate the following initial graph:

- The item \mathcal{I} as well as all items of `TABLE` constitute the vertices,
- There is an arc from \mathcal{I} to each item of the `TABLE` collection.

We provide the following mechanism for deriving new collections:

- In a first phase we declare the name of the new collection as well as the names of its attributes and their respective types. This is achieved exactly in the same way as those collections that are used in the arguments of a global constraint (see page 4).

EXAMPLE: Consider again the example of the `element(INDEX, TABLE, VALUE)` constraint. The declaration `ITEM - collection(index - dvar, value - dvar)` introduces a new collection called `ITEM` where each item has an `index` and a `value` attribute. Both attributes correspond to domain variables.

- In a second phase we give a list of patterns that are used for generating the items of the new collection. A pattern `o - item(a1 - v1, a2 - v2, ..., an - vn)` or `item(a1 - v1, a2 - v2, ..., an - vn)` specifies for each attribute `ai` ($1 \leq i \leq n$) of the new collection how to fill it⁷. This is done by providing for each attribute `ai` one of the following element `vi`:

⁷`o` is one of the comparison operators `=`, `≠`, `<`, `≥`, `>`, `≤`. When omitted its default value is `=`.

- A constant,
- A parameter of the global constraint G ,
- An attribute of a collection that is a parameter of the global constraint G ,
- An attribute of a derived collection that was previously declared.

This element v_i must be compatible with the type declaration of the corresponding attribute of the new collection.

EXAMPLE: We continue the example of the `element(INDEX, TABLE, VALUE)` constraint and the derived collection `ITEM – collection(index – dvar, value – dvar)`. The pattern `item(index – INDEX, value – VALUE)` indicates that:

- The `index` attribute of the `ITEM` collection will be generated by using the `INDEX` argument of the `element` constraint. Since `INDEX` is a domain variable, it is compatible with the declaration `ITEM – collection(index – dvar, value – dvar)` of the new collection.
- The `value` attribute of the `ITEM` collection will be generated by using the `VALUE` argument of the `element` constraint. `VALUE` is also compatible with the declaration statement of the new collection.

We now describe how we use the pattern for generating the items of a derived collection. We have the following two cases:

- If the pattern $o - \text{item}(a_1 - v_1, a_2 - v_2, \dots, a_n - v_n)$ does not contain any reference to an attribute of a collection then we generate one single item for such pattern⁸. In this context the value v_i of the attribute a_i ($1 \leq i \leq n$) corresponds to a constant, to an argument of the global constraint or to a new derived collection.
- If the pattern $o - \text{item}(a_1 - v_1, a_2 - v_2, \dots, a_n - v_n)$, where o is one of the comparison operators $=, \neq, <, \geq, >, \leq$, contains one or several references to an attribute of a collection⁹ we denote by:
 - k_1, k_2, \dots, k_m the indices of the positions corresponding to the attribute of a collection within $\text{item}(a_1 - v_1, a_2 - v_2, \dots, a_n - v_n)$,
 - $c_{\alpha_1}, c_{\alpha_2}, \dots, c_{\alpha_m}$ the corresponding collections,
 - $a_{\alpha_1}, a_{\alpha_2}, \dots, a_{\alpha_m}$ the corresponding attributes.

For each combination of items $c_{\alpha_1}[i_1], c_{\alpha_2}[i_2], \dots, c_{\alpha_m}[i_m]$ such that:

$$i_1 \in [1, |c_{\alpha_1}|], i_2 \in [1, |c_{\alpha_2}|], \dots, i_m \in [1, |c_{\alpha_m}|] \text{ and } i_1 \ o \ i_2 \ o \ \dots \ o \ i_n$$

we generate an item of the new derived collection $(a_1 - w_1 \ a_2 - w_2 \ \dots \ a_n - w_n)$ defined by:

$$w_j (1 \leq j \leq n) = \begin{cases} c_{\alpha_p}[i_p].a_{\alpha_p} & \text{if } j \in \{k_1, k_2, \dots, k_m\}, j = k_p \\ v_j & \text{if } j \notin \{k_1, k_2, \dots, k_m\} \end{cases} .$$

⁸In this first case the value of o is irrelevant.

⁹This collection is a parameter of the global constraint or corresponds to a newly derived collection.

1.2. DESCRIBING GLOBAL CONSTRAINTS IN TERMS OF GRAPH PROPERTIES 19

We illustrate this generation process on a set of examples. Each example is described by providing:

- The global constraint and its arguments,
- The declaration of the new derived collection,
- The pattern used for creating an item of the new collection,
- The items generated by applying this pattern to the global constraint,
- A comment about the generation process.

We first start with four examples that don't mention any references to an attribute of a collection. A box surrounds an argument of a global constraint that is mentioned in a generated item.

EXAMPLE

```
CONSTRAINT      : element(INDEX, TABLE, VALUE)
DERIVED COLLECTION: ITEM - collection(index - dvar, value - dvar)
PATTERN(S)      : item(index - INDEX, value - VALUE)
GENERATED ITEM(S) : {index - INDEX value - VALUE}
```

We generate one single item where the two attributes `index` and `value` respectively take the first argument `INDEX` and the third argument `VALUE` of the `element` constraint.

EXAMPLE

```
CONSTRAINT      : lex_lesseq(VECTOR1, VECTOR2)
DERIVED COLLECTION: DESTINATION - collection(index - int, x - int, y - int)
PATTERN(S)      : item(index - 0, x - 0, y - 0)
GENERATED ITEM(S) : {index - 0 x - 0 y - 0}
```

We generate one single item where the three attributes `index`, `x` and `y` take value 0.

EXAMPLE

```
CONSTRAINT      : in_relation(VARIABLES, TUPLES_OF_VALS)
DERIVED COLLECTION: TUPLES_OF_VARS - collection(vec - TUPLE_OF_VARS)
PATTERN(S)      : item(vec - VARIABLES)
GENERATED ITEM(S) : {vec - VARIABLES}
```

We generate one single item where the unique attribute `vec` takes the first argument of the `in_relation` constraint as its value.

EXAMPLE

```

CONSTRAINT      : domain_constraint(VAR, VALUES)
DERIVED COLLECTION: VALUE – collection(var01 – int, value – dvar)
PATTERN(S)      : item(var01 – 1, value – VAR)
GENERATED ITEM(S) : {var01 – 1 value – VAR}

```

We generate one single item where the two attributes var01 and value respectively take value 1 and the first argument of the domain_constraint constraint.

We continue with three examples that mention one or several references to an attribute of some collections. We now need to explicitly give the items of these collections in order to generate the items of the derived collection.

EXAMPLE

```

CONSTRAINT      : lex_lesseq(VECTOR1, VECTOR2)
VECTOR1         : {var – 5, var – 2, var – 3, var – 1}
VECTOR2         : {var – 5, var – 2, var – 6, var – 2}
DERIVED COLLECTION: COMPONENTS – collection(index – int,
                                             x – dvar, y – dvar)
PATTERN(S)      : item(index – VECTOR1.keya,
                       x – VECTOR1.var, y – VECTOR2.var)
GENERATED ITEM(S) : {index – 1 x – 5 y – 5, index – 2 x – 2 y – 2,
                    index – 3 x – 3 y – 6, index – 4 x – 1 y – 2}

```

The pattern mentions three references VECTOR1.key, VECTOR1.var and VECTOR2.var to the collections VECTOR1 and VECTOR2 used in the arguments of the lex_lesseq constraint. $\forall i_1 \in [1, |\text{VECTOR1}|], \forall i_2 \in [1, |\text{VECTOR2}|]$ such that $i_1 = i_2$ ^b we generate an item $\text{index} - v_1 \ x - v_2 \ y - v_3$ where:

$$v_1 = i_1, \quad v_2 = \text{VECTOR1}[i_1].\text{var}, \quad v_3 = \text{VECTOR2}[i_1].\text{var}.$$

This leads to the four items listed in the GENERATED ITEM(S) field.

^aAs defined in Section 1.1.2 page 4, key is an implicit attribute corresponding to the position of an item within a collection.

^bWe use an equality since this is the default value of the comparison operator o when we don't use a pattern of the form $o - \text{item}(\dots)$.

EXAMPLE

```

CONSTRAINT      : cumulatives(TASKS, MACHINES, CTR)
TASKS           : {machine - 1 origin - 1 duration - 4 end - 5 height - 1,
                  machine - 1 origin - 4 duration - 2 end - 6 height - 3,
                  machine - 1 origin - 2 duration - 3 end - 5 height - 2,
                  machine - 2 origin - 5 duration - 2 end - 7 height - 2}
DERIVED COLLECTION: TIME_POINTS - collection(idm - int,
                                             duration - dvar, point - dvar)
PATTERN(S)     : item(idm - TASKS.machine,
                      duration - TASKS.duration, point - TASKS.origin)
                : item(idm - TASKS.machine,
                      duration - TASKS.duration, point - TASKS.end)
GENERATED ITEM(S) : {idm - 1 duration - 4 point - 1,
                    idm - 1 duration - 2 point - 4,
                    idm - 1 duration - 3 point - 2,
                    idm - 2 duration - 2 point - 5,
                    idm - 1 duration - 4 point - 5,
                    idm - 1 duration - 2 point - 6,
                    idm - 1 duration - 3 point - 5,
                    idm - 2 duration - 2 point - 7}

```

The two patterns mention the references `TASKS.machine`, `TASKS.duration`, `TASKS.origin` and `TASKS.end` of the `TASKS` collection used in the arguments of the `cumulatives` constraint. $\forall i \in [1, |\text{TASKS}|]$, we generate two items `idm - u1 duration - u2 point - u3`, `idm - v1 duration - v2 point - v3` where:

$$\begin{aligned}
 u_1 &= \text{TASKS}[i].\text{machine}, & u_2 &= \text{TASKS}[i].\text{duration}, & u_3 &= \text{TASKS}[i].\text{origin}, \\
 v_1 &= \text{TASKS}[i].\text{machine}, & v_2 &= \text{TASKS}[i].\text{duration}, & v_3 &= \text{TASKS}[i].\text{end}.
 \end{aligned}$$

This leads to the eight items listed in the `GENERATED ITEM(S)` field.

EXAMPLE

```

CONSTRAINT      : golomb(VARIABLES)
VARIABLES      : {var - 0, var - 1, var - 4, var - 6}
DERIVED COLLECTION: PAIRS - collection(x - dvar, y - dvar)
PATTERN(S)     : > -item(x - VARIABLES.var, y - VARIABLES.var)
GENERATED ITEM(S) : {x - 1 y - 0,
                    x - 4 y - 0, x - 4 y - 1,
                    x - 6 y - 0, x - 6 y - 1, x - 6 y - 4}

```

The pattern mentions two references `VARIABLES.var` and `VARIABLES.var` to the `VARIABLES` collection used in the arguments of the `golomb` constraint. $\forall i_1 \in [1, |\text{VARIABLES}|], \forall i_2 \in [1, |\text{VARIABLES}|]$ such that $i_1 > i_2^a$ we generate the item $x - u_1 y - u_2$ where:

$$u_1 = \text{VARIABLES}[i_1].\text{var}, \quad u_2 = \text{VARIABLES}[i_2].\text{var}.$$

This leads to the six items listed in the `GENERATED ITEM(S)` field.

^aWe use the comparison operator `>` since we have a pattern of the form `> -item(...)`.

Elementary constraints attached to the arcs

This section describes the constraints that are associated with the arcs of the initial graph of a global constraint. These constraints are called *arc constraints*. To each arc one can associate one or several arc constraints. An arc will belong to the final graph if and only if all its arc constraints hold. An arc constraint from a vertex v_1 to a vertex v_2 mentions variables and/or values associated with v_1 and v_2 . Before defining an *arc constraint*, we first need to introduce *simple arithmetic expressions* as well as *arithmetic expressions*. Simple arithmetic expressions and arithmetic expressions are defined recursively.

Simple arithmetic expressions A *simple arithmetic expression* is defined by one of the five following expressions.

- *I*: *I* is an integer.
- *Arg*: *Arg* is an argument of the global constraint of type `int` or `dvar`.
- *Arg*: *Arg* is a formal parameter provided by the arc generator¹⁰ of the graph-constraint.
- *Col.Attr*: *Col* is a formal parameter provided by the arc generator or the collection used in the `For all items of iterator`¹¹. *Attr* is an attribute of the collection referenced by *Col*.

¹⁰Arc generators are described in Section 1.2.2 (page 26).

¹¹The `For all items of iterator` is described in Section 1.2.3 (page 43).

EXAMPLE: As an example consider the first graph-constraint associated with the `global_cardinality_with_costs(VARIABLES, VALUES, MATRIX, COST)` constraint and its arc constraint `variables.var = VALUES.val`. Both, `variables.var` as well as `VALUES.val` are *simple arithmetic expressions* of the form `Col.Attr`:

- In `variables.var`, `variables` corresponds to the formal parameter provided by the arc generator $SELF \mapsto \text{collection}(\text{variables})$, while `var` is an attribute of the `VARIABLES` collection.
- In `VALUES.val`, `VALUES` corresponds to the collection denoted by the `For` all items of iterator, while `val` is an attribute of the `VALUES` collection.

- `Col[Expr].Attr`: `Col` is an argument of type collection, `Attr` one attribute of `Col` and `Expr` an *arithmetic expression*.

`Col[Expr].Attr` denotes the value of attribute `Attr` of the `Exprth` item of the collection denoted by `Col`.

EXAMPLE: As an example consider the `global_cardinality_with_costs(VARIABLES, VALUES, MATRIX, COST)` constraint and its second graph-constraint, which defines the `COST` variable. The expression `MATRIX[(variables.key - 1) * |VALUES| + values.key].c` is a *simple arithmetic expression* of the form `Col[Expr].Attr`:

- `MATRIX` is a collection of items `collection(i - int, j - int, c - int)` where all items are sorted in increasing order on attributes `i, j` (because of the restriction `increasing_seq(MATRIX, [i, j])`).
- `MATRIX[(variables.key - 1) * |VALUES| + values.key].c` denotes the value of attribute `c` of an item of the `MATRIX` collection. The position of this item within the `MATRIX` collection depends on the position of a variable of the `VARIABLES` collection^a as well as on the position of a value of the `VALUES` collection^b.

^aThis position is denoted by the expression `variables.key`. As defined in Section 1.1.2 page 4, `key` is an implicit attribute corresponding to the position of an item within a collection.

^bThis position is denoted by the expression `values.key`.

Arithmetic expressions An *arithmetic expression* is recursively defined by one of the following expressions:

- A *simple arithmetic expression*.
- `Exp1 Op Exp2`:
 - `Exp1` is an *arithmetic expression*,
 - `Op` is one of the following symbols `+`, `-`, `*`, `/`¹²,
 - `Exp2` is an *arithmetic expression*.
- `|Collection|`:
 - `Collection` is an argument of type collection and `|Collection|` denotes the number of items of that collection.

¹²/ _{denotes an integer division, a division in which the fractional part is discarded.}

- $|\text{Exp}|$:
 - Exp is an *arithmetic expression*, and $|\text{Exp}|$ denotes the absolute value of this expression.
- $\text{sign}(\text{Exp})$:
 - Exp is an *arithmetic expression*, and $\text{sign}(\text{Exp})$ the sign of Exp (-1 if Exp is negative, 0 if Exp is equal to 0 , 1 if Exp is positive).

EXAMPLE: An example of use of `sign` can be found in the last part of the arc constraint of the crossing constraint:

$$\text{sign}((s2.ox - s1.ex) * (s1.ey - s1.oy) - (s1.ex - s1.ox) * (s2.oy - s1.ey)) \neq \text{sign}((s2.ex - s1.ex) * (s2.oy - s1.oy) - (s2.ox - s1.ox) * (s2.ey - s1.ey))$$

- $\text{card_set}(\text{Set})$:
 - Set is a reference to a set of integers or to a set variable. $\text{card_set}(\text{Set})$ denotes the number of elements of that set.

EXAMPLE: An example of use of `card_set` can be found in the `symmetric_gcc` constraint: `vars.nocc = card_set(vars.var)`.

- $\text{SimpleExp}_1 \bmod \text{SimpleExp}_2$,
 $\min(\text{SimpleExp}_1, \text{SimpleExp}_2)$ or $\max(\text{SimpleExp}_1, \text{SimpleExp}_2)$:
 - SimpleExp_1 is a *simple arithmetic expression*,
 - SimpleExp_2 is a *simple arithmetic expression*.

Arc constraints Now that we have introduced *simple arithmetic expressions* as well as *arithmetic expressions* we define an *arc constraint*. An *arc constraint* is recursively defined by one of the following expressions:

- TRUE:

This stands for an arc constraint that always holds. As a result, the corresponding arc always belongs to the final graph.

EXAMPLE: An example of use of TRUE can be found in the `sum_ctr(VARIABLES, CTR, VAR)` constraint, where it is used in order to enforce keeping all items of the VARIABLES collection in the final graph.

- $\text{Exp}_1 \text{ Comparison } \text{Exp}_2$:
 - Exp_1 is an *arithmetic expression*,
 - Comparison is one of the comparison operators $\leq, \geq, <, >, =, \neq$,
 - Exp_2 is an *arithmetic expression*.

EXAMPLE: As an example of such arc constraint, the second graph-constraint of the `cumulative(TASKS, LIMIT)` constraint uses the following arc constraints:

- `tasks1.duration > 0`,
- `tasks2.origin ≤ tasks1.origin`,
- `tasks1.origin < tasks2.end`.

The conjunction of these three arc constraints can be interpreted in the following way: An arc from a task `tasks1` to a task `tasks2` will belong to the final graph if and only if `tasks2` overlaps the origin of `tasks1`.

- `Exp1 SimpleCtr Exp2`:
 - `Exp1` is an *arithmetic expression*,
 - `SimpleCtr` is an argument of type `atom` that can only take one of the values `≤, ≥, <, >, =, ≠`,
 - `Exp2` is an *arithmetic expression*.

EXAMPLE: An example of use of such an arc constraint can be found in the `change(NCHANGE, VARIABLES, CTR)` constraint: `variables1.var CTR variables2.var`. Within this expression, `variables1` and `variables2` correspond to consecutive items of the `VARIABLES` collection.

- `Exp1 ¬SimpleCtr Exp2`:
 - `Exp1` is an *arithmetic expression*,
 - `SimpleCtr` is an argument of type `atom` that can only take one of the values `≤, ≥, <, >, =, ≠`,
 - `Exp2` is an *arithmetic expression*.

EXAMPLE: An example of use of such an arc constraint can be found in the `change_continuity(NB_PERIOD_CHANGE, NB_PERIOD_CONTINUITY, MIN_SIZE_CHANGE, MAX_SIZE_CHANGE, MIN_SIZE_CONTINUITY, MAX_SIZE_CONTINUITY, NB_CHANGE, NB_CONTINUITY, VARIABLES, CTR)` constraint: `variables1.var ¬CTR variables2.var`. Within this expression, `variables1` and `variables2` correspond to consecutive items of the `VARIABLES` collection.

- `Ctr(Exp1, ..., Expn)`:
 - `Ctr` is a global constraint defined in the catalog for which there exists a graph-based and/or an automaton-based representation,
 - `Exp1, ..., Expn` correspond to the arguments of the global constraint `Ctr`. Each argument should be a *simple arithmetic expression* that is compatible with the type declaration of the argument of `Ctr`.

EXAMPLE: An example of such arc constraint can be found in the definition of `diffn`: `diffn(ORTHOTOPE1, ORTHOTOPE2)` uses the `two_orth_do_not_overlap(ORTHOTOPE1, ORTHOTOPE2)` global constraint for defining its arc constraint. Since `ORTHOTOPE1` and `ORTHOTOPE2` correspond to items of `ORTHOTOPE1` and `ORTHOTOPE2` respectively and since both `ORTHOTOPE1` and `ORTHOTOPE2` correspond to items of `ORTHOTOPE1` there is no type compatibility problem between the call to `two_orth_do_not_overlap` and its definition.

- `ArcCtr1 LogicalConnector ArcCtr2`:
 - `ArcCtr1` is an *arc constraint*,
 - `LogicalConnector` is one of the logical connectors \vee , \wedge , \Rightarrow , \Leftrightarrow ,
 - `ArcCtr2` is an *arc constraint*.

EXAMPLE: As shown by the following example, `minimum(MIN, VARIABLES)` uses this kind of arc constraint: `variables1 = variables2 \vee variables1.var < variables2.var`, where `variables1` and `variables2` correspond to items of the `VARIABLES` collection, holds if and only if one of the following conditions holds:

- `variables1` and `variables2` correspond to the same item of the `VARIABLES` collection,
- The `var` attribute of `variables1` is strictly less than the `var` attribute of `variables2`.

Graph generators

This section describes how to generate the initial graph associated with a global constraint. Initial graphs correspond to directed hypergraphs [29], which have a very regular structure. They are defined in the following way:

- The vertices of the directed hypergraph are generated from collections of items such that each item corresponds to one vertex of the directed hypergraph. These collections are either collections that arise as arguments of the global constraint, or collections that are derived from one or several arguments of the global constraint. In this latter case these *derived collections* are computed by using the *collection generators* previously introduced (see Section 1.2.2, page 17).
- To all arcs of the directed hypergraph corresponds the same arc constraint that involves vertices in a given order¹³. These arc constraints, which are mainly unary and binary constraints, were described in the previous section (see Section 1.2.2, page 22). We describe all the arcs of an initial graph with a set of predefined *arc generators*, which correspond to classical regular structures one can find in the graph literature [30, pages 140–153]. An *arc generator* of arity a

¹³Usually the edges of a hypergraph are not oriented [29, pages 1–2]. However for our purpose we need to define an order on the vertices of an edge since the corresponding arc constraint takes its arguments in a given order.

1.2. DESCRIBING GLOBAL CONSTRAINTS IN TERMS OF GRAPH PROPERTIES 27

takes n collections of items, denoted $c_i (1 \leq i \leq n)$, as input and returns the corresponding hypergraph where the vertices are the items of the input collections $c_i (1 \leq i \leq n)$ and where all arcs involve a vertices. Specific arc generators allow for giving an a -ary constraint for which a is not fixed, which means that the corresponding hypergraph contains arcs involving various number of vertices.

Each arc generator has a name and takes one or several collections of items as input and generates a set of arcs. Each arc is made from a sequence of items $i_1 i_2 \dots i_a$ and is denoted by (i_1, i_2, \dots, i_a) . a is called the *arity* of the arc generator. We have the following types of arc generators:

- Arc generators with a fixed predefined arity. In fact most arc generators have a fixed predefined arity of 2. The graphs they generate correspond to digraphs.
- Arc generators that can be used with any arity a greater than or equal to 1. These arc generators generate directed hypergraphs where all arcs consist of a items.
- Arc generators that generate arcs that don't involve the same number of items.

We now give the list of arc generators, listed in alphabetic order, and the arcs they generate. For each arc generator we point to a global constraint where it is used in practice. Finally, Figure 1.4 illustrates the different arc generators. At present the following arc generators are in use:

- *CHAIN* has a predefined arity of 2. It takes one collection c and generates the following arcs¹⁴:

$$- \forall i \in [1, |c| - 1]: (c[i], c[i + 1]), \quad - \forall i \in [1, |c| - 1]: (c[i + 1], c[i]).$$

EXAMPLE: The arc generator *CHAIN* is for instance used in the `group_skip_isolated_item` constraint.

- *CIRCUIT* has a predefined arity of 2. It takes one collection c and generates the following arcs:

$$- \forall i \in [1, |c| - 1]: (c[i], c[i + 1]), \quad - (c[|c|], c[1]).$$

EXAMPLE: The arc generator *CIRCUIT* is for instance used in the `circular_change` constraint.

- *CLIQUE* can be used with any arity a greater than or equal to 2. It takes one collection c and generates the arcs: $\forall i_1 \in [1, |c|], \forall i_2 \in [1, |c|], \dots, \forall i_a \in [1, |c|] : (c[i_1], c[i_2], \dots, c[i_a])$.

EXAMPLE: The arc generator *CLIQUE* is usually used with an arity $a = 2$. This is for instance the case of the `alldifferent` constraint.

¹⁴As defined in Section 1.1.2 (page 4) we use the following notation: For a given collection c , $|c|$ and $c[i]$ respectively denote the number of items of c and the i^{th} item of c .

- *CLIQUE*(Comparison) , where Comparison is one of the comparison operators $\leq, \geq, <, >, =, \neq$, can be used with any arity a greater than or equal to 2. It takes one collection c and generates the arcs:

$$\begin{aligned} \forall i_1 \in [1, |c|], \\ \forall i_2 \in [1, |c|] \text{ such that } i_1 \text{ Comparison } i_2, \\ \dots\dots\dots, \\ \forall i_a \in [1, |c|] \text{ such that } i_{a-1} \text{ Comparison } i_a : (c[i_1], c[i_2], \dots, c[i_a]). \end{aligned}$$

EXAMPLE: The `orchard(TREES)` constraint is an example of constraint that uses the *CLIQUE*($<$) arc generator with an arity $a = 3$. It generates an arc for each set of three trees.

- *GRID*($[d_1, d_2, \dots, d_n]$) takes a collection c consisting of $d_1 \cdot d_2 \cdot \dots \cdot d_n$ items and generates the arcs $(c[i], c[j])$ where i and j satisfy the following condition. There exists a natural number α ($0 \leq \alpha \leq n - 1$) such that (1) and (2) hold:

$$\begin{aligned} (1) |i - j| &= \prod_{1 \leq k \leq \alpha} d_k \text{ (when } \alpha = 0 \text{ we have } \prod_{1 \leq k \leq \alpha} d_k = 1), \\ (2) \lfloor \frac{i}{\prod_{1 \leq k \leq \alpha+1} d_k} \rfloor &= \lfloor \frac{j}{\prod_{1 \leq k \leq \alpha+1} d_k} \rfloor. \end{aligned}$$

EXAMPLE: The `connect_points` constraint uses the *GRID* arc generator.

- *LOOP* has a predefined arity of 2. It takes one collection c and generates the arcs: $\forall i \in [1, |c|]: (c[i], c[i])$. *LOOP* is usually used in order to generate a loop on some vertices, so that they don't disappear from the final graph.

EXAMPLE: The `global_contiguity(VARIABLES)` constraint is an example of constraint that uses the *LOOP* arc generator so that each variable of the `VARIABLES` collection belongs to the final graph.

- *PATH* can be used with any arity a greater than or equal to 1. It takes one collection c , and generates the following arcs: $\forall i \in [1, |c| - a + 1] : (c[i], c[i + 1], \dots, c[i + a - 1])$.

EXAMPLE: *PATH* is for instance used in the `sliding_sum(LOW, UP, SEQ, VARIABLES)` constraint with an arity `SEQ`, where `SEQ` is an argument of the `sliding_sum` constraint.

- *PATH_1* generates arcs that don't involve the same number of items. It takes one collection c , and generates the following arcs: $(c[1]), (c[1], c[2]), \dots, (c[1], c[2], \dots, c[|c|])$.

EXAMPLE: *PATH_1* is used in the `size_maximal_starting_sequence_alldifferent` constraint.

1.2. DESCRIBING GLOBAL CONSTRAINTS IN TERMS OF GRAPH PROPERTIES 29

- *PATH_N* generates arcs that don't involve the same number of items. It takes one collection c , and generates the following arcs: $\forall i \in [1, |c|], \forall j \in [i, |c|] : (c[i], c[i+1], \dots, c[j])$.

EXAMPLE: *PATH_N* is for instance used in the `size_maximal_sequence_alldifferent` constraint.

- *PRODUCT* has a predefined arity of 2. It takes two collections c_1, c_2 and generates the arcs: $\forall i \in [1, |c_1|], \forall j \in [1, |c_2|] : (c_1[i], c_2[j])$.

EXAMPLE: *PRODUCT* is for instance used in the `same(VARIABLES1, VARIABLES2)` constraint for generating an arc from every item of the `VARIABLES1` collection to every item of the `VARIABLES2` collection.

- *PRODUCT(Comparison)*, where *Comparison* is one of the comparison operators $\leq, \geq, <, >, =, \neq$, has a predefined arity of 2. It takes two collections c_1, c_2 and generates the arcs: $\forall i \in [1, |c_1|], \forall j \in [1, |c_2|]$ such that $i \text{ Comparison } j : (c_1[i], c_2[j])$.

EXAMPLE: *PRODUCT(=)* is for instance used in the `differ_from_at_least_k_pos(K, VECTOR1, VECTOR2)` constraint in order to generate an arc between the i^{th} component of `VECTOR1` and the i^{th} component of `VECTOR2`.

- *SELF* has a predefined arity of 1. It takes one collection c and generates the arcs: $\forall i \in [1, |c|] : (c[i])$.

EXAMPLE: *SELF* is for instance used in the `among(NVAR, VARIABLES, VALUES)` constraint in order to generate a unary arc constraint `in(variables.var, VALUES)` for each variable of the `VARIABLES` collection.

- *SYMMETRIC_PRODUCT* has a predefined arity of 2. It takes two collections c_1, c_2 and generates the following arcs: $\forall i \in [1, |c_1|], \forall j \in [1, |c_2|] : (c_1[i], c_2[j])$ and $(c_2[j], c_1[i])$. *SYMMETRIC_PRODUCT* is currently not used.

- *SYMMETRIC_PRODUCT(Comparison)*, where *Comparison* is one of the comparison operators $\leq, \geq, <, >, =, \neq$, has a predefined arity of 2. It takes two collections c_1, c_2 and generates the arcs: $\forall i \in [1, |c_1|], \forall j \in [1, |c_2|]$ such that $i \text{ Comparison } j : (c_1[i], c_2[j])$ and $(c_2[j], c_1[i])$.

EXAMPLE: The `two_orth_do_not_overlap` constraint is an example of constraint that uses the *SYMMETRIC_PRODUCT(=)* arc generator.

- *VOID* takes one collection and does not generate any arc.

EXAMPLE: *VOID* is for instance used in the `lex_lesseq` constraint.

Finally, we can combine the *PRODUCT* arc generator with the arc generators from the following set $Generator = \{CIRCUIT, CHAIN, CLIQUE, LOOP, PATH, VOID\}$. This is achieved by using the construction $PRODUCT(G_1, G_2)$ where G_1 and G_2 belong to $Generator$. It applies G_1 to the first collection c_1 passed to *PRODUCT* and G_2 to the second collection c_2 passed to *PRODUCT*. Finally, it applies *PRODUCT* on c_1 and c_2 . In a similar way the $PRODUCT(Comparison)$ arc generator is extended to $PRODUCT(G_1, G_2, Comparison)$.

EXAMPLE: As an illustrative example, consider the `alldifferent_same_value(NSAME, VARIABLES1, VARIABLES2)` constraint, which uses the arc generator $PRODUCT(CLIQUE, LOOP, =)$ on the collections `VARIABLES1` and `VARIABLES2`. It generates the following arcs:

- Since the first argument of *PRODUCT* is *CLIQUE* it generates an arc between each pair of items of the `VARIABLES1` collection.
- Since the second argument of *PRODUCT* is *LOOP* it generates a loop for each item of the `VARIABLES2` collection.
- Since the third argument is the comparison operator `=` it finally generates an arc between an item of the `VARIABLES1` collection and an item of the `VARIABLES2` collection when the two items have the same position.

Figure 1.3 shows the generated graph under the hypothesis that `VARIABLES1` and `VARIABLES2` have respectively 3 and 3 items.

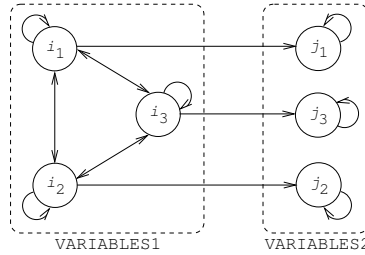


Figure 1.3: Example of initial graph generated by $PRODUCT(CLIQUE, LOOP, =)$

Figure 1.4 illustrates the different arc generators. On the one hand, for those arc generators that take one single collection, we apply them on the collection of items $\{i - 1, i - 2, i - 3, i - 4\}$. On the other hand, for those arc generators that take two collections, we apply them on $\{i - 1, i - 2\}$ and $\{i - 3, i - 4\}$. We use the following pictogram for the graphical representation of a constraint network:

- A line for an arc constraint of arity 1,
- An arrow for an arc constraint of arity 2,
- A closed line for an arc constraint with an arity strictly greater than 2. In this last case, since the vertices of an arc are ordered, a black circle at one of the extremities indicates the direction of the closed line. For instance consider the

1.2. DESCRIBING GLOBAL CONSTRAINTS IN TERMS OF GRAPH PROPERTIES 31

example of *PATH_1* in Figure 1.4. The closed line that contains vertices 1, 2 and 3 means that a 3-ary arc constraint involves items 1, 2, and 3 in this specific order.

Dotted circles represent vertices that don't belong to the graph. This stems from the fact that the arc generator did not produce any arc involving these vertices. The leftmost lowest corner indicates the arity of the corresponding arc generator:

- An integer if it has a fixed predefined arity,
- n if it can be used with any arity greater than or equal to 1,
- $*$ if it generates arcs that don't necessarily involve the same number of items.

Graph properties

We represent a global constraint as the search of a subgraph (i.e. a final graph) of a known initial graph, so that this final graph satisfies a given set of graph properties. Most graph properties have the form $\text{Char Comparison Exp}$ or the form $\text{Char} \notin [\text{Exp}_1, \text{Exp}_2]$, where *Char* is a graph characteristic [17], [31], *Comparison* is one of the comparison operators $=, <, \geq, >, \leq, \neq$, and *Exp*, *Exp*₁, *Exp*₂ are expressions that can be evaluated to an integer. Before defining each graph characteristic, let's first introduce some basic vocabulary on graphs.

Graph terminology and notations A *digraph* $G = (V(G), E(G))$ is a pair where $V(G)$ is a finite set, called the set of *vertices*, and where $E(G)$ is a set of ordered pairs of vertices, called the set of *arcs*. The *arc*, *path*, *circuit* and *strongly connected component* of a graph G correspond to oriented concepts, while the *edge*, *chain*, *cycle* and *connected component* are non-oriented concepts. However, as reported in [17, page 6] an undirected graph can be seen as a digraph where to each edge we associate the corresponding two arcs. Parts (A) and (B) of Figure 1.5 respectively illustrate the terms for undirected graphs and digraphs.

- We say that e_2 is a *successor* of e_1 if there exists an arc that starts from e_1 and ends at e_2 . In the same way, we say that e_2 is a *predecessor* of e_1 if there exists an arc that starts from e_2 and ends at e_1 .
- A vertex of G that does not have any predecessor is called a *source*. A vertex of G that does not have any successor is called a *sink*.
- A sequence (e_1, e_2, \dots, e_k) of edges of G such that each edge has a common vertex with the previous edge, and the other vertex common to the next edge is called a *chain* of length k . A chain where all vertices are distinct is called an *elementary chain*. Each equivalence class of the relation " e_i is equal to e_j " or there exists a chain between e_i and e_j " is a *connected component* of the graph G .

CIRCUIT 2	LOOP 1	PRODUCT (<->) 2
CHAIN 2	PATH n	PRODUCT (PATH, VOID) 2
CLIQUE n	PATH_1 *	SELF 1
CLIQUE (<=) 2	PATH_N *	SYMMETRIC_PRODUCT 2
CLIQUE (>=) 2	PRODUCT 2	SYMMETRIC_PRODUCT (<=) 2
CLIQUE (<) 2	PRODUCT (<=) 2	SYMMETRIC_PRODUCT (>=) 2
CLIQUE (>) 2	PRODUCT (>=) 2	SYMMETRIC_PRODUCT (<) 2
CLIQUE (<->) 2	PRODUCT (<) 2	SYMMETRIC_PRODUCT (>) 2
GRID ([2, 2]) 2	PRODUCT (>) 2	SYMMETRIC_PRODUCT (=) 2
CYCLE 2	PRODUCT (=) 2	SYMMETRIC_PRODUCT (<->) 2

Figure 1.4: Examples of arc generators

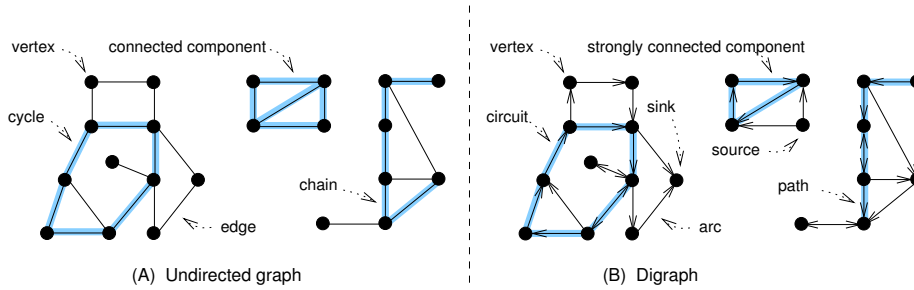


Figure 1.5: Graph terminology for an undirected graph and a digraph

- A sequence (e_1, e_2, \dots, e_k) of arcs of G such that for each arc e_i ($1 \leq i < k$) the end of e_i is equal to the start of the arc e_{i+1} is called a *path* of length k . A path where all vertices are distinct is called an *elementary path*. Each equivalence class of the relation " e_i is equal to e_j or there exists a path between e_i and e_j " is a *strongly connected component* of the graph G .
- A chain (e_1, e_2, \dots, e_k) of G is called a *cycle* if the same edge does not occur more than once in the chain and if the two extremities of the chain coincide. A cycle (e_1, e_2, \dots, e_k) of G is called a *circuit* if for each edge e_i ($1 \leq i < k$), the end of e_i is equal to the start of the edge e_{i+1} .
- Given a graph G , we define the *reduced graph* $R(G)$ of G as follows: To each strongly connected component of G corresponds a vertex of $R(G)$. To each arc of G that connects different strongly connected components corresponds an arc in $R(G)$.
- The *rank* function associated with the vertices $V(G)$ of a graph G that does not contain any circuit is defined in the following way:
 - The rank of the vertices that do not have any predecessor (i.e. the sources) is equal to 0,
 - The rank r of a vertex v that is not a source is the length of longest path (e_1, e_2, \dots, e_r) such that the start of the arc e_1 is a source and the end of arc e_r is the vertex v .

We now present the different notations used in the catalog:

- $[k]$ corresponds to $\{1, \dots, k\}$ for k any positive integer.
- Given a set X , $|X|$ is the number of its elements.
- Given two sets X and Y , $X \uplus Y$ denotes the union of the two sets when they are disjoint.
- Given a digraph G and $x \in V(G)$, $d_G^+(x) = |\{y : y \in V(G) : (x, y) \in E(G)\}|$ and $d_G^-(x) = |\{y : y \in V(G) : (y, x) \in E(G)\}|$.

- Given a digraph G and X a subset of $V(G)$, the subdigraph of G induced by X is the digraph $G[X]$ where $V(G[X]) = X$ and $E(G[X]) = X^2 \cap E(G)$. By aim of simplicity, we denote $G[V(G) - X]$ by $G - X$. Moreover, if $X = \{x\}$, we use $G - x$ instead of $G - \{x\}$.
- Given two digraph G_1 and G_2 such that $V(G_1) \cap V(G_2) = \emptyset$, $G_1 \oplus G_2$ denotes the graph whose vertices set is $V(G_1) \cup V(G_2)$ and whose arcs set is $E(G_1) \cup E(G_2)$.
- Given a graph characteristic $\mathbf{CH} \in \{\mathbf{NCC}, \mathbf{NSCC}\}$, a digraph G and an integer k , $\mathbf{CH}(G, k)$ is the number of connected components (respectively strongly connected components) of G with cardinal k .

Given a graph characteristics, for instance the number of connected components, $\mathbf{NCC}_{\text{INITIAL}}$ will denote the number of connected components of the initial graph (i.e. the graph induced by the constraint under consideration), \mathbf{NCC} will denote the number of connected components of the final graph (i.e. a subgraph of the initial graph). The use of $\mathbf{NCC}(G)$ will denote the number of connected components of the digraph G .

Given a global constraint C , and a graph characteristics \mathbf{GC} used in the description of C , $\underline{\mathbf{GC}}$ (resp. $\overline{\mathbf{GC}}$) denotes a lower bound (resp. upper bound) of \mathbf{GC} among all possible final graphs compatible with the current status of C .

Graph characteristics We list in alphabetic order the different *graph characteristics* we consider for a final graph $G_f = (V(G_f), E(G_f))$ associated with a global constraint and give an example of constraint where they are used:

- **MAX_DRG** : largest distance between sources and sinks in the reduced graph associated with G_f (adjacent vertices are at a distance of 1).

EXAMPLE: We don't provide any example since **MAX_DRG** is currently not used.

- **MAX_ID** : number of predecessors of the vertex of G_f that has the maximum number of predecessors without counting an arc from a vertex to itself.

EXAMPLE: The `circuit` constraint uses the graph property **MAX_ID** = 1 in order to force each vertex of the final graph to have at most one predecessor.

- **MAX_NCC** : number of vertices of the largest connected component of G_f .

EXAMPLE: The `longest_change(SIZE, VARIABLES, CTR)` constraint uses the graph property **MAX_NCC** = **SIZE** in order to catch in **SIZE** the maximum number of consecutive variables of the **VARIABLES** collection for which constraint **CTR** holds.

1.2. DESCRIBING GLOBAL CONSTRAINTS IN TERMS OF GRAPH PROPERTIES³⁵

- **MAX_NSICC** : number of vertices of the largest strongly connected component of G_f .

EXAMPLE: The `tree` constraint covers a digraph by a set of trees in such a way that each vertex belongs to a distinct tree. It uses the graph-property $\text{MAX_NSICC} \leq 1$ in order to avoid to have any circuit involving more than one vertex.

- **MAX_OD** : number of successors of the vertex of G_f that has the maximum number of successors without counting an arc from a vertex to itself.

EXAMPLE: The `tour` constraint enforces to cover a graph with a Hamiltonian cycle. It uses the graph-property $\text{MAX_OD} = 2$ to enforce that each vertex of G_f have at most two^a successors.

^aSince the `tour` constraint uses the $\text{CLIQUE}(\neq)$ arc generator the vertices of G_f don't have any loop.

- **MIN_DRG** : smallest distance between sources and sinks in the reduced graph associated with G_f (adjacent vertices are at a distance of 1).

EXAMPLE: We don't provide any example since **MIN_DRG** is currently not used by any constraint.

- **MIN_ID** : number of predecessors of the vertex of G_f that has the minimum number of predecessors without counting an arc from a vertex to itself.

EXAMPLE: The `tour` constraint enforces to cover a graph with a Hamiltonian cycle. It uses the graph-property $\text{MIN_ID} = 2$ to enforce that each vertex of G_f have at most two^a predecessors.

^aSince the `tour` constraint uses the $\text{CLIQUE}(\neq)$ arc generator the vertices of G_f don't have any loop.

- **MIN_NCC** : number of vertices of the smallest connected component of G_f .

EXAMPLE: Within the `group` constraint, each connected component of G_f corresponds to a maximum sequence of consecutive variables that take their value in a given set of values. Therefore, the graph-property $\text{MIN_NCC} = \text{MIN_SIZE}$ enforces that the smallest sequence of such variables consist of MIN_SIZE variables.

- **MIN_NSICC** : number of vertices of the smallest strongly connected component of G_f .

EXAMPLE: The `circuit(NODES)` constraint enforces covering a digraph with one circuit visiting once all its vertices. The graph-property `MIN_NSICC = |NODES|` enforces that the smallest strongly connected component of G_f contain `|NODES|` vertices. Since `|NODES|` also corresponds to the number of vertices of the initial graph this means that G_f is a strongly connected component involving all the vertices. This is clearly a necessary condition^a for having a circuit visiting once all vertices.

^aOf course, this is not enough, and the description of the `circuit` constraint asks for some other properties.

- **MIN_OD** : number of successors of the vertex of G_f that has the minimum number of successors without counting an arc from a vertex to itself.

EXAMPLE: The `tour` constraint enforces to cover a graph with a Hamiltonian cycle. It uses the graph-property `MIN_OD = 2` to enforce that each vertex of G_f have at most two^a successors.

^aSince the `tour` constraint uses the `CLIQUE(≠)` arc generator the vertices of G_f don't have any loop.

- **NARC** : cardinality of the set $E(G_f)$.

EXAMPLE: The `disjoint(VARIABLES1, VARIABLES2)` constraint enforces that each variable of the collection `VARIABLES1` take a value that is distinct from all the values assigned to the variables of the collection `VARIABLES2`.

This is imposed by creating an arc from each variable of `VARIABLES1` to each variable of `VARIABLES2`. To each arc corresponds an equality constraint involving the variables associated to the extremities of the arc. Finally, the graph property `NARC = 0` forces G_f to be empty so that no value is both assigned to a variable of `VARIABLES1` as well as to a variable of `VARIABLES2`.

- **NARC_NO_LOOP** : cardinality of the set $E(G_f)$ without considering the arcs linking the same vertex (i.e. a loop).

EXAMPLE: The constraint `alldifferent_same_value` uses the `NARC_NO_LOOP` graph-property.

1.2. DESCRIBING GLOBAL CONSTRAINTS IN TERMS OF GRAPH PROPERTIES 37

- **NCC** : number of connected components of G_f .

EXAMPLE: The `tree` constraint covers a digraph by NTREES trees in such a way that each vertex belongs to a distinct tree. It uses the graph-property $NCC = NTREES$ in order to state that G_f is made up from NTREES connected components.

- **NSCC** : number of strongly connected components of G_f .

EXAMPLE: The constraint `nvalue(NVAL, VARIABLES)` forces NVAL to be equal to the number of distinct values assigned to the variables of the collection VARIABLES. This is enforced by using the graph-property $NSCC = NVAL$. Each strongly connected component of the final graph corresponds to the variables that are assigned to the same value.

- **NSINK** : number of vertices of G_f that do not have any successor.

EXAMPLE: The `same(VARIABLES1, VARIABLES2)` enforces that the variables of the VARIABLES1 collection correspond to the variables of the VARIABLES2 collection according to a permutation. We first create an arc from each variable of VARIABLES1 to each variable of VARIABLES2. To each arc corresponds an equality constraint involving the variables associated with the extremities of the arc. We use the graph-property $NSINK = |VARIABLES2|$ in order to express the fact that each value assigned to a variable of VARIABLES2 should also be assigned to a variable of VARIABLES1.

- **NSINK_NSOURCE** : sum over the different connected components of G_f of the minimum of the number of sinks and the number of sources of a connected component.

EXAMPLE: The `soft_same_var(C, VARIABLES1, VARIABLES2)` constraint enforces C to be the minimum number of values to change in the VARIABLES1 and the VARIABLES2 collections of variables^a, so that the variables of VARIABLES2 correspond to the variables of VARIABLES1 according to a permutation. A connected component C_{val} of the final graph G_f corresponds to all variables that are assigned to the same value val : the sources and the sinks of C_{val} respectively correspond to the variables of VARIABLES1 and to the variables of VARIABLES2 that are assigned to val . For a connected component, the minimum of the number of sources and sinks expresses the number of variables for which we don't need to make any change. Therefore we use the graph-property $NSINK_NSOURCE = |VARIABLES1| - C$ for encoding the meaning of the `soft_same_var` constraint.

^aBoth collections have the same number of variables.

- **NSOURCE** : number of vertices of G_f that do not have any predecessor.

EXAMPLE: The `same(VARIABLES1, VARIABLES2)` enforces that the variables of the `VARIABLES1` collection correspond to the variables of the `VARIABLES2` collection according to a permutation.

We first create an arc from each variable of `VARIABLES1` to each variable of `VARIABLES2`. To each arc corresponds an equality constraint involving the variables associated with the extremities of the arc. We use the graph-property `NSOURCE = |VARIABLES1|` in order to express the fact that each value assigned to a variable of `VARIABLES1` should also be assigned to a variable of `VARIABLES2`.

- **NTREE** : number of vertices of G_f that do not belong to any circuit and for which at least one successor belongs to a circuit. Such vertices can be interpreted as root nodes of a tree.

EXAMPLE: The `cycle(NCYCLE, NODES)` enforces that `NCYCLE` equal the number of circuits for covering an initial graph in such a way that each vertex belongs to one single circuit.

The graph-property `NTREE = 0` enforces that all vertices of the final graph belong to a circuit.

- **NVERTEX** : cardinality of the set $V(G_f)$.

EXAMPLE: The `cutset(SIZE_CUTSET, NODES)` constraint considers a digraph with n vertices described by the `NODES` collection. It enforces that the subset of kept vertices of cardinality $n - \text{SIZE_CUTSET}$ and their corresponding arcs form a graph without a circuit. It uses the graph-property `NVERTEX = n - SIZE_CUTSET` for enforcing that the final graph G_f contain the required number of vertices.

- **RANGE_DRG** : difference between the largest distance between sources and sinks in the reduced graph associated with G_f and the smallest distance between sources and sinks in the reduced graph associated with G_f .

EXAMPLE: The `tree_range` constraint enforces to cover a digraph in such a way that each vertex belongs to a distinct tree. In addition it forces the difference between the longest and the shortest paths of G_f to be equal to the variable `R`. For this purpose it uses the graph-property `RANGE_DRG = R`.

1.2. DESCRIBING GLOBAL CONSTRAINTS IN TERMS OF GRAPH PROPERTIES 39

- **RANGE_NCC** : difference between the number of vertices of the largest connected component of G_f and the number of vertices of the smallest connected component of G_f .

EXAMPLE: We don't provide any example since **RANGE_NCC** is currently not used by any constraint.

- **RANGE_NSCC** : difference between the number of vertices of the largest strongly connected component of G_f and the number of vertices of the smallest strongly connected component of G_f .

EXAMPLE: The `balance(BALANCE, VARIABLES)` constraint forces **BALANCE** to be equal to the difference between the number of occurrence of the value that occurs the most and the value that occurs the least within the collection of variables **VARIABLES**. Each strongly connected component of G_f corresponds to the variables that are assigned to the same value. The graph property **RANGE_NSCC = BALANCE** allows for expressing this definition.

- **ORDER**(rank, default, attr) :
 - rank is an integer or an argument of type integer of the global constraint,
 - default is an integer,
 - attr is an attribute corresponding to an integer or to a domain variable that occurs in all the collections that were used for generating the vertices of the initial graph.

We explain what is the value associated with **ORDER**(rank, default, attr). Let \mathcal{V} denote the vertices of rank rank of G_f from which we remove any loops.

- When \mathcal{V} is not empty, it corresponds to the values of attribute attr of the items associated with the vertices of \mathcal{V} ,
- Otherwise, when \mathcal{V} is empty, it corresponds to the default value default.

EXAMPLE: The `minimum(MIN, VARIABLES)` forces **MIN** to be the minimum value of the collection of domain variables **VARIABLES**. There is an arc from a variable var_1 to a variable var_2 if and only if $\text{var}_1 < \text{var}_2$. The graph-property **ORDER**(0, MAXINT, var) = **MIN** expresses the fact that **MIN** is equal to the value of the source of G_f (since rank = 0).

- **PATH_FROM_TO**(attr, from, to) :

- `attr` is an attribute corresponding to an integer or to a domain variable that occurs in all the collections that were used for generating the vertices of the initial graph,
- `from` is an integer or an argument of type integer of the global constraint,
- `to` is an integer or an argument of type integer of the global constraint.

Let \mathcal{F} (respectively \mathcal{T}) denote the vertices of G_f such that `attr` is equal to `from` (respectively `to`). `PATH_FROM_TO(attr, from, to)` is equal to 1 if there exists a path between each vertex of \mathcal{F} and each vertex of \mathcal{T} , and 0 otherwise.

EXAMPLE: The constraint `lex_lesseq` uses the `PATH_FROM_TO` graph-property.

• **PRODUCT**(`col`, `attr`)

- `col` is a collection that was used for generating the vertices of the initial graph,
- `attr` is an attribute corresponding to an integer or to a domain variable of the collection `col`.

Let \mathcal{V} be the set of vertices of G_f that were generated from the items of the collection `col`.

- If \mathcal{V} is not empty, `PRODUCT(col, attr)` corresponds to the product of the values of attribute `attr` associated with the vertices of \mathcal{V} ,
- Otherwise, if \mathcal{V} is empty, `PRODUCT(col, attr)` is equal to 1.

EXAMPLE: The constraint `product_ctr(VARIABLES, CTR, VAR)` forces the product of the variables of the `VARIABLES` collection to be equal, less than or equal, ... to a given domain variable `VAR`.

To each variable of `VARIABLES` corresponds a vertex of the initial graph. Since we want to keep all the vertices of the initial graph we use the `SELF` arc generator together with the `TRUE` arc constraint. Finally, `PRODUCT(VARIABLES, var) CTR VAR` expresses the required condition. In this expression `var` and `CTR` respectively corresponds to the attribute of the collection `VARIABLES` (a domain variable) and to the condition we want to enforce. Since the final graph G_f contains all the vertices of the initial graph, the expression `PRODUCT(VARIABLES, var)` corresponds to the product of the variables of the `VARIABLES` collection.

• **RANGE**(`col`, `attr`) :

- `col` is a collection that was used for generating the vertices of the initial graph,

1.2. DESCRIBING GLOBAL CONSTRAINTS IN TERMS OF GRAPH PROPERTIES 41

- `attr` is an attribute corresponding to an integer or to a domain variable of the collection `col`.

Let \mathcal{V} be the set of vertices of G_f that were generated from the items of the collection `col`.

- If \mathcal{V} is not empty, $\mathbf{RANGE}(\text{col}, \text{attr})$ corresponds to the difference between the maximum and the minimum values of attribute `attr` associated with the vertices of \mathcal{V} ,
- Otherwise, if \mathcal{V} is empty, $\mathbf{RANGE}(\text{col}, \text{attr})$ is equal to 0.

EXAMPLE: The constraint `range_ctr(VARIABLES, CTR, VAR)` forces the difference between the maximum value and the minimum value of the variables of the `VARIABLES` collection to be equal, less than or equal, ... to a given domain variable `VAR`. To each variable of `VARIABLES` corresponds a vertex of the initial graph. Since we want to keep all the vertices of the initial graph we use the `SELF` arc generator together with the `TRUE` arc constraint. Finally, $\mathbf{RANGE}(\text{VARIABLES}, \text{var})$ `CTR` `VAR` expresses the required condition. In this expression `var` and `CTR` respectively corresponds to the attribute of the collection `VARIABLES` (a domain variable) and to the condition we want to enforce. Since the final graph G_f contains all the vertices of the initial graph, the expression $\mathbf{RANGE}(\text{VARIABLES}, \text{var})$ corresponds to the difference between the maximum value and the minimum value of the variables of the `VARIABLES` collection.

- $\mathbf{SUM}(\text{col}, \text{attr})$:
 - `col` is a collection that was used for generating the vertices of the initial graph,
 - `attr` is an attribute corresponding to an integer or to a domain variable of the collection `col`.

Let \mathcal{V} be the set of vertices of G_f that were generated from the items of the collection `col`.

- If \mathcal{V} is not empty, $\mathbf{SUM}(\text{col}, \text{attr})$ corresponds to the sum of the values of attribute `attr` associated with the vertices of \mathcal{V} ,
- Otherwise, if \mathcal{V} is empty, $\mathbf{SUM}(\text{col}, \text{attr})$ is equal to 0.

EXAMPLE: The constraint `sum_ctr(VARIABLES, CTR, VAR)` forces the sum of the variables of the `VARIABLES` collection to be equal, less than or equal, ... to a given domain variable `VAR`.

To each variable of `VARIABLES` corresponds a vertex of the initial graph. Since we want to keep all the vertices of the initial graph we use the *SELF* arc generator together with the *TRUE* arc constraint. Finally, `SUM(VARIABLES, var) CTR VAR` expresses the required condition. In this expression `var` and `CTR` respectively correspond to the attribute of the collection `VARIABLES` (a domain variable) and to the condition we want to enforce. Since the final graph G_f contains all the vertices of the initial graph, the expression `SUM(VARIABLES, var)` corresponds to the sum of the variables of the `VARIABLES` collection.

- **SUM_WEIGHT_ARC(Expr)** : `Expr` is an *arithmetic expression*. For each arc a of $E(G_f)$, let $f(a)$ denote the value of `Expr`. **SUM_WEIGHT_ARC(Expr)** is equal to $\sum_{a \in E(G_f)} f(a)$. The value of `Expr` usually depends on the attributes of the items located at the extremities of an arc.

EXAMPLE: The constraint `global_cardinality_with_costs(VARIABLES, VALUES, MATRIX, COST)` enforces that each value `VALUES[i].val` be assigned to exactly `VALUES[i].noccurrence` variables of the `VARIABLES` collection. In addition the `COST` of an assignment is equal to the sum of the elementary costs associated with the fact that we assign the i^{th} variable of the `VARIABLES` collection to the j^{th} value of the `VALUES` collection. These elementary costs are given by the `MATRIX` collection.

The graph-property `SUM_WEIGHT_ARC(MATRIX[(variables.key-1)*size(VALUES)+values.key].c) = COST` expresses the fact that the `COST` variable is equal to the sum of the elementary costs associated with each variable-value assignment. All these elementary costs are recorded in the `MATRIX` collection. More precisely, the cost c_{ij} is recorded in the attribute `c` of the $((i-1) * |VALUES| + j)^{th}$ entry of the `MATRIX` collection.

A last graph characteristic, **DISTANCE**, is computed on two final graphs G_1 and G_2 that have the same set V of vertices and the sets $E(G_1)$ and $E(G_2)$ of arcs. This graph characteristic is the cardinality of the set $(E(G_1) - E(G_2)) \cup (E(G_2) - E(G_1))$. This corresponds to the number of arcs that belong to $E(G_1)$ but not to $E(G_2)$, plus the number of arcs that are in $E(G_2)$ but not in $E(G_1)$.

1.2.3 Graph constraint

A global constraint can be defined as a conjunction of several *simple* or *dynamic graph constraints*¹⁵ that all share the same name, the same arguments and the same argument restrictions¹⁶. This section first describes *simple graph constraints* and then *dynamic graph constraints*, which are an extension of *simple graph constraints*.

¹⁵For an example of global constraint that is defined by more than one graph constraint see for instance the *sort* constraint and its two graph constraints.

¹⁶The arguments and the argument restrictions were described in Section 1.1.4, page 13.

Simple graph constraint

To a *simple graph constraint* correspond several initial graphs, usually one, where all the initial graphs have the same vertices and arcs. Specifying more than one initial graph is achieved by using the FOR ALL ITEMS OF iterator, which takes a collection C and generates an initial graph $G_i(t)$ for each item t of C . In this context, the arc constraints and/or graph properties of an initial graph may depend of the attributes of the item t of C from which they were generated. All arc constraints attached to a given arc¹⁷ have to be pairwise mutually incompatible¹⁸.

The graphs of a *simple graph constraint* are defined by the following fields:

- An **Arc input(s)** field, which consists of a sequence of collections C_1, C_2, \dots, C_d ($d \geq 1$). To each item of these collections corresponds a vertex of the initial graph.
- An **Arc generator** field, which can be one or several expressions¹⁹ of the following forms:

- $ARC_GENERATOR \mapsto \text{collection}(\text{item}_1, \text{item}_2, \dots, \text{item}_a)$, where $ARC_GENERATOR$ is one of the arc generators with a fixed arity²⁰ defined in Section 1.2.2 page 26, and item_i ($1 \leq i \leq a$) denotes the i^{th} item associated with the i^{th} vertex of an arc. These items correspond to formal parameters²¹ which can be used within an arc constraint. When the **Arc input(s)** field consists of one single collection ($d = 1$), item_i ($1 \leq i \leq a$) represents an item of the collection C_1 . Otherwise, when $d > 1$, we must have $a = d$ and, in this context, item_i ($1 \leq i \leq a$) represents an item of C_i .

EXAMPLE: The `alldifferent(VARIABLES)` constraint has the following **Arc input(s)** and **Arc generator** fields:

- * Its **Arc input(s)** field refers only to the collection `VARIABLES` (i.e. $d = 1$).
- * Its **Arc generator** field consists of
 $CLIQUE \mapsto \text{collection}(\text{variables1}, \text{variables2})$ (i.e. $a = 2$).

In this context, where $d = 1$, both `variables1` and `variables2` are items of the `VARIABLES` collection.

¹⁷As we previously said, even if we have more than one initial graph, all vertices and arcs of the different initial graphs are identical.

¹⁸Two arc constraints $\text{ctr}_1(X_1, X_2, \dots, X_n)$ and $\text{ctr}_2(X_1, X_2, \dots, X_n)$ are *incompatible* if there does not exist any tuple of values $\langle v_1, v_2, \dots, v_n \rangle$ such that both $\text{ctr}_1(X_1, X_2, \dots, X_n)$ and $\text{ctr}_2(X_1, X_2, \dots, X_n)$ hold.

¹⁹Usually one single expression.

²⁰Any arc generator different from `PATH_1` and `PATH_N`.

²¹See the description of *simple arithmetic expressions* page 22.

EXAMPLE: The same(VARIABLES1, VARIABLES2) constraint has the following **Arc input(s)** and **Arc generator** fields:

- * Its **Arc input(s)** field refers to the collections VARIABLES1 and VARIABLES2 (i.e. $d = 2$).
- * Its **Arc generator** field consists of
 $PRODUCT \mapsto \text{collection}(\text{variables1}, \text{variables2})$ (i.e. $a = 2$).

In this context, where $d > 1$, variables1 and variables1 respectively correspond to items of the VARIABLES1 and the VARIABLES2 collections.

- $ARC_GENERATOR \mapsto \text{collection}$, where $ARC_GENERATOR$ is one of the arc generators $PATH_1$ or $PATH_N$. In this context, collection denotes a collection of items corresponding to the vertices of an arc of the initial graph. An arc constraint enforces a restriction on the items of this collection.

EXAMPLE:

The `size_maximal_sequence_alldifferent` (SIZE, VARIABLES) constraint has the following **Arc input(s)** and **Arc generator** fields:

- * Its **Arc input(s)** field refers to the VARIABLES collection.
- * Its **Arc generator** field consists of $PRODUCT \mapsto \text{collection}$.

In this context, collection is a collection of the same type as the VARIABLES collection. It corresponds to the variables associated with an arc of the initial graph.

When the **Arc generator** field consists of n ($n > 1$) expressions then these expressions have the form:

$$ARC_GENERATOR_1 \mapsto \text{collection}(\text{item}_1, \text{item}_2, \dots, \text{item}_a)$$

$$ARC_GENERATOR_2 \mapsto \text{collection}(\text{item}_1, \text{item}_2, \dots, \text{item}_a)$$

.....

$$ARC_GENERATOR_n \mapsto \text{collection}(\text{item}_1, \text{item}_2, \dots, \text{item}_a)$$

All leftmost part of the expressions must be the same since they will be involved in one single **Arc constraint(s)** field. The `global_contiguity` constraint is an example of global constraint where more than one arc generator is used.

- An **Arc arity** field, which corresponds to the number of vertices a of each arc of the initial graph. a is either a strictly positive integer, an argument of the global constraint of type `int`, or the character `*`. In this last case, this is used for denoting the fact that all the arc constraints don't involve the same number of vertices. This is for instance the case when we use the arc generators $PATH_1$ or $PATH_N$ as in the `arith_sliding` or the `size_maximal_sequence_alldifferent` constraints.
- An **Arc constraint(s)** field, which corresponds to a conjunction of *arc constraints*²² those were introduced in Section 1.2.2 page 22.

²²Usually this conjunction consists of one single *arc constraint*.

1.2. DESCRIBING GLOBAL CONSTRAINTS IN TERMS OF GRAPH PROPERTIES 45

- A **Graph property(ies)** field, which corresponds to one or several *graph properties* (see Section 1.2.2 page 31) to be satisfied on the final graphs associated with an instantiated solution of the global constraint. To each initial graph corresponds one final graph obtained by removing all arcs for which the corresponding arc constraints do not hold as well as all vertices that don't have any arc.

We now give several examples of descriptions of *simple graph constraints*, starting from the `nvalue` constraint, which was introduced as a first example of global constraint that can be modeled by a graph property in Section 1.2.1 page 14.

EXAMPLE: The constraint `nvalue(NVAL, VARIABLES)` restricts `NVAL` to be the number of distinct values taken by the variables of the collection `VARIABLES`. Its meaning is described by a *simple graph constraint* corresponding to the following items:

Arc input(s) : `VARIABLES`
Arc generator : `CLIQUE` \mapsto `collection(variables1, variables2)`
Arc arity : 2
Arc constraint(s) : `variables1.var = variables2.var`
Graph property(ies): `NSCC = NVAL`

Since this description does not use the `FOR ALL ITEMS OF` iterator we generate one single initial graph. Each vertex of this graph corresponds to one item of the `VARIABLES` collection. Since we use the `CLIQUE` arc generator we have an arc between each pair of vertices. An arc constraint corresponds to an equality constraint between the two variables that are associated with the extremities of the arc. Finally, the **Graph property(ies)** field forces the final graph to have `NVAL` strongly connected components.

EXAMPLE: The constraint `global_contiguity(VARIABLES)` forces all variables of the `VARIABLES` collection to be assigned to 0 or 1. In addition, all variables assigned to value 1 appear contiguously. Its meaning is described by a *simple graph constraint* corresponding to the following items:

Arc input(s) : `VARIABLES`
Arc generator : `PATH` \mapsto `collection(variables1, variables2)`
 `LOOP` \mapsto `collection(variables1, variables2)`
Arc arity : 2
Arc constraint(s) : `variables1.var = variables2.var`
 `variables1.var = 1`
Graph property(ies): `NCC \leq 1`

Since this description does not use the `FOR ALL ITEMS OF` iterator we generate one single initial graph. Each vertex of this graph corresponds to one item of the `VARIABLES` collection. Since we use the `PATH` arc generator we generate an arc from item `VARIABLES[i]` to item `VARIABLES[i + 1]` ($1 \leq i < |\text{VARIABLES}|$). In addition, since we use the `LOOP` arc generator, we generate also an arc from each item of the `VARIABLES` collection to itself⁹. The effect of the arc constraint is to keep in the final graph those vertices for which the corresponding variable is assigned to 1. Adjacent variables assigned to 1 form a connected component of the final graph and the graph property `NCC \leq 1` enforces to have at most one such group of adjacent variables assigned to 1.

⁹We use the `LOOP` arc generator in order to keep in the final graph those isolated variables assigned to 1. This is because isolated vertices with no arcs are always removed from the final graph.

EXAMPLE:

The `global_cardinality(VARIABLES, VALUES)` constraint enforces that each value `VALUES[i].val` ($1 \leq i \leq |\text{VALUES}|$) be taken by exactly `VALUES[i].noccurrence` variables of the `VARIABLES` collection. Its meaning is described by a *simple graph constraint* corresponding to the following items:

For all items of VALUES:

Arc input(s) : `VARIABLES`
Arc generator : `SELF` \mapsto `collection(variables)`
Arc arity : 1
Arc constraint(s) : `variables.var = VALUES.val`
Graph property(ies): `NVERTEX = VALUES.noccurrence`

Since this description uses the **For all items of VALUES** iterator on the `VALUES` collection we generate an initial graph for each item of the `VALUES` collection (i.e. one graph for each value). Each vertex of an initial graph corresponds to one item of the `VARIABLES` collection. Since we use the `SELF` arc generator we have an arc for each vertex. For an initial graph associated with a value `val` an arc constraint on a vertex `v` corresponds to an equality constraint between the variable associated with `v` and the value `val`. Finally, the **Graph property(ies)** field forces the final graph to have a given number of vertices (i.e. associated with the attribute `val`).

Dynamic graph constraint

The purpose of a *dynamic graph constraint* is to enforce a condition on different subsets of variables, not known in advance. This situation occurs frequently in practice and is hard to express since one cannot use a classical constraint for which it is required to provide all variables right from the beginning. One good example of such global constraint is the *cumulative* constraint where one wants to force the sum of some variables to be less than or equal to a given limit. In the context of the *cumulative* constraint, each set of variables is defined by the height of the different tasks that overlap a given instant i . Since the origins of the tasks are not initially fixed, we don't know in advance which task will overlap a given instant and so, we cannot state any sum constraint initially.

A *dynamic graph constraint* is defined in exactly the same way as a *simple graph constraint*, except that we may omit the **Graph property(ies)** field, and that we have to provide the two following additional fields:

- The **Set** field denotes a generator of sets of vertices. Such a generator takes as argument a final graph and produces different sets of vertices. In order to have something tractable, we force the total number of generated sets to be polynomial in the number of vertices.

In practice each set of vertices is represented by a collection of items. The type of this collection corresponds either to the type of the items associated with the vertices, or to the type of a new derived collection. This is achieved by providing an expression of the form `name` or `name-derived_collection`, where `name` represents a formal parameter, and `derived_collection` a declaration of a new derived collection (as specified in Section 1.2.2, page 17).

- The **Constraint(s) on sets** field provides a global constraint defined in the catalog that has to hold for each set created by the previous generator.

We now describe the different generators of sets of vertices currently available:

- `ALL_VERTICES` generates one single set containing all the vertices of the final graph. It is specified by a declaration of the form

```
ALL_VERTICES >> [vertices]
```

where `vertices` represents all the vertices of the final graph.

- `CC` generates one set of vertices for each connected component of the final graph. These sets correspond to all the vertices of a given connected component. It is specified by a declaration of the form

```
CC >> [connected_component]
```

where `connected_component` represents the vertices of a connected component of the final graph.

- `PATH_LENGTH(L)` generates all elementary paths²³ of L vertices of the final graph such that, discarding loops, all vertices of a path have no more than one successor and one predecessor in the final graph. It is specified by a declaration of the form

`PATH_LENGTH(L)>> [path]`

where `path` represents the vertices of an elementary path, ordered according to their occurrence in the path.

- `PRED` generates the non-empty sets corresponding to the predecessors of each vertex of the final graph. It is specified by a declaration of the form

`PRED>> [predecessor, destination]`

where `destination` represents a vertex of the final graph and `predecessor` its predecessors.

- `SUCC` generates the non-empty sets corresponding to the successors of each vertex of the final graph. It is specified by a declaration of the form

`SUCC>> [source, successor]`

where `source` represents a vertex of the final graph and `successor` its successors.

As an illustrative example of *dynamic graph constraint* we now consider the *cumulative constraint*.

²³A path where all vertices are distinct is called an *elementary path*.

EXAMPLE: The `cumulative(TASKS,LIMIT)` constraint, where `TASKS` is a collection of the form `collection(origin - dvar,duration - dvar,end - dvar,height - dvar)`, and where `LIMIT` is a non-negative integer, holds if, for any point the cumulated height of the set of tasks that overlap that point, does not exceed `LIMIT`.

The first graph constraint of `cumulative` enforces for each task of the `TASKS` collection the equality `origin + duration = end`. We focus on the second graph constraint, which uses a *dynamic graph constraint* described by the following items:

```

Arc input(s)      : TASKS TASKS
Arc generator    : PRODUCT  $\mapsto$  collection(tasks1, tasks2)
Arc arity        : 2
Arc constraint(s) : tasks1.duration > 0
                    tasks2.origin  $\leq$  tasks1.origin
                    tasks1.origin  $\leq$  tasks2.end
Sets             : SUCC>>
                    [source,
                     variables - col(VARIABLES - collection(var - dvar),
                                       [item(var - TASKS.height)])]
Constraint(s) on sets: sum_ctr(variables,  $\leq$ , LIMIT)

```

The second graph constraint is defined by:

- To each item of the `TASKS` collection correspond two vertices of the initial graph.
- The arity of the arc constraint is 2.
- The arcs of the initial graph are constructed with the *PRODUCT* arc generator between the `TASKS` collection and the `TASKS` collection. Therefore, each vertex associated with a task is linked to all the vertices related to the different tasks.
- The arc constraint that is associated with an arc between a task `tasks1` and a task `tasks2` is an overlapping constraint that holds if both, the duration of `tasks1` is strictly greater than zero, and if the origin of `tasks1` is overlapped by task `tasks2`.
- The set generator is `SUCC`. The final graph will consist of those tasks for which the origin is covered by at least one task and of those corresponding tasks.
- The dynamic constraint on a set forces the sum of the heights of the tasks that belong to a successor set to not exceed `LIMIT`.

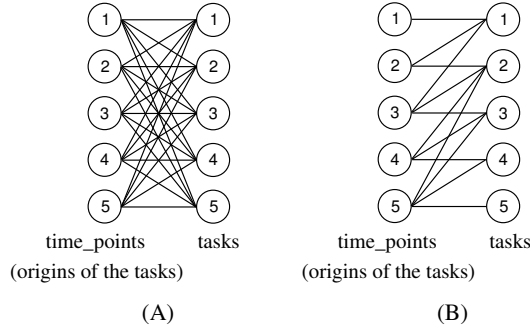


Figure 1.6: Initial and final graph of an instance of the cumulative constraint

Parts (A) and (B) of Figure 1.6 respectively show the initial and the final graph corresponding to the following instance:

```
cumulative({origin - 1 duration - 3 height - 1,
            origin - 2 duration - 9 height - 2,
            origin - 3 duration - 10 height - 1,
            origin - 6 duration - 6 height - 1,
            origin - 7 duration - 2 height - 3}, 8).
```

We label the vertices of the initial and final graph by giving the key^a of the corresponding task. On both graphs the edges are oriented from left to right. On the final graph we consider the sets that consist of the successors of the different vertices; those are the sets of tasks {1}, {1, 2}, {1, 2, 3}, {2, 3, 4} and {2, 3, 4, 5}. Since the SUCC set generator uses a derived collection that only considers the height attribute of a task, these sets respectively correspond to the following collection of items:

- {var - 1},
- {var - 1, var - 2},
- {var - 1, var - 2, var - 1},
- {var - 2, var - 1, var - 1},
- {var - 2, var - 1, var - 1, var - 3}.

The cumulative constraint holds since, for each successors set, the corresponding constraint holds:

- $\text{sum_ctr}(\{\text{var} - 1\}, \leq, 8)$,
- $\text{sum_ctr}(\{\text{var} - 1, \text{var} - 2\}, \leq, 8)$,
- $\text{sum_ctr}(\{\text{var} - 1, \text{var} - 2, \text{var} - 1\}, \leq, 8)$,
- $\text{sum_ctr}(\{\text{var} - 2, \text{var} - 1, \text{var} - 1\}, \leq, 8)$,
- $\text{sum_ctr}(\{\text{var} - 2, \text{var} - 1, \text{var} - 1, \text{var} - 3\}, \leq, 8)$.

The $\text{sum_ctr}(\text{VARIABLES}, \text{CTR}, \text{VAR})$ constraint holds if the sum S of the variables of the VARIABLES collection satisfies $S \text{ CTR } \text{VAR}$, where CTR is a comparison operator.

^akey is an implicit attribute corresponding to the position of an item within a collection that was introduced in Section 1.1.2, page 4.

1.3 Describing global constraints in terms of automata

This section is based on the paper describing global constraint in terms of automata [4]. The main difference with the original paper is the introduction of array of counters within the description of an automaton. We consider global constraints for which any ground instance can be checked in linear time by scanning once through their variables without using any data structure, except counters or arrays of counters. In order to concretely illustrate this point we first select a set of global constraints and write down a checker for each of them. Finally, we give for each checker a sketch of the corresponding automaton. Based on these observations, we define the type of automaton we use in the catalog.

1.3.1 Selecting an appropriate description

As we previously said, we focus on those global constraints that can be checked by scanning once through their variables. This is for instance the case of:

- `element` [32],
- `lex_lesseq` [36],
- `minimum` [33],
- `among` [37],
- `pattern` [34],
- `inflexion` [3],
- `global_contiguity` [35],
- `alldifferent` [18].

Since they illustrate key points needed for characterizing the set of solutions associated with a global constraint, our discussion will be based on the last five constraints for which we now recall the definition:

- The `global_contiguity(vars)` constraint forces the sequence of 0-1 variables `vars` to have at most one group of consecutive 1. For instance, the constraint `global_contiguity([0, 1, 1, 0])` holds since we have only one group of consecutive 1.
- The lexicographic ordering constraint $\vec{x} \leq_{\text{lex}} \vec{y}$ (see `lex_lesseq`) over two vectors of variables $\vec{x} = \langle x_0, \dots, x_{n-1} \rangle$ and $\vec{y} = \langle y_0, \dots, y_{n-1} \rangle$ holds iff $n = 0$ or $x_0 < y_0$ or $x_0 = y_0$ and $\langle x_1, \dots, x_{n-1} \rangle \leq_{\text{lex}} \langle y_1, \dots, y_{n-1} \rangle$.
- The `among(nvar, vars, values)` constraint restricts the number of variables of the sequence of variables `vars` that take their value in a given set `values`, to be equal to the variable `nvar`. For instance, `among(3, [4, 5, 5, 4, 1], [1, 5, 8])` holds since exactly 3 values of the sequence 45541 are located in $\{1, 5, 8\}$.
- The `inflexion(ninf, vars)` constraint forces the number of inflexions of the sequence of variables `vars` to be equal to the variable `ninf`. An *inflexion* is described by one of the following patterns: a strict increase followed by a strict decrease or, conversely, a strict decrease followed by a strict increase. For instance, `inflexion(4, [3, 3, 1, 4, 5, 5, 6, 5, 5, 6, 3])` holds since we can extract from the

sequence 33145565563 the four subsequences 314, 565, 6556 and 563, which all follow one of these two patterns.

- The `alldifferent(vars)` constraint forces all pairs of distinct variables of the collection `vars` to take distinct values. For instance `alldifferent([6, 1, 5, 9])` holds since we have four distinct values.

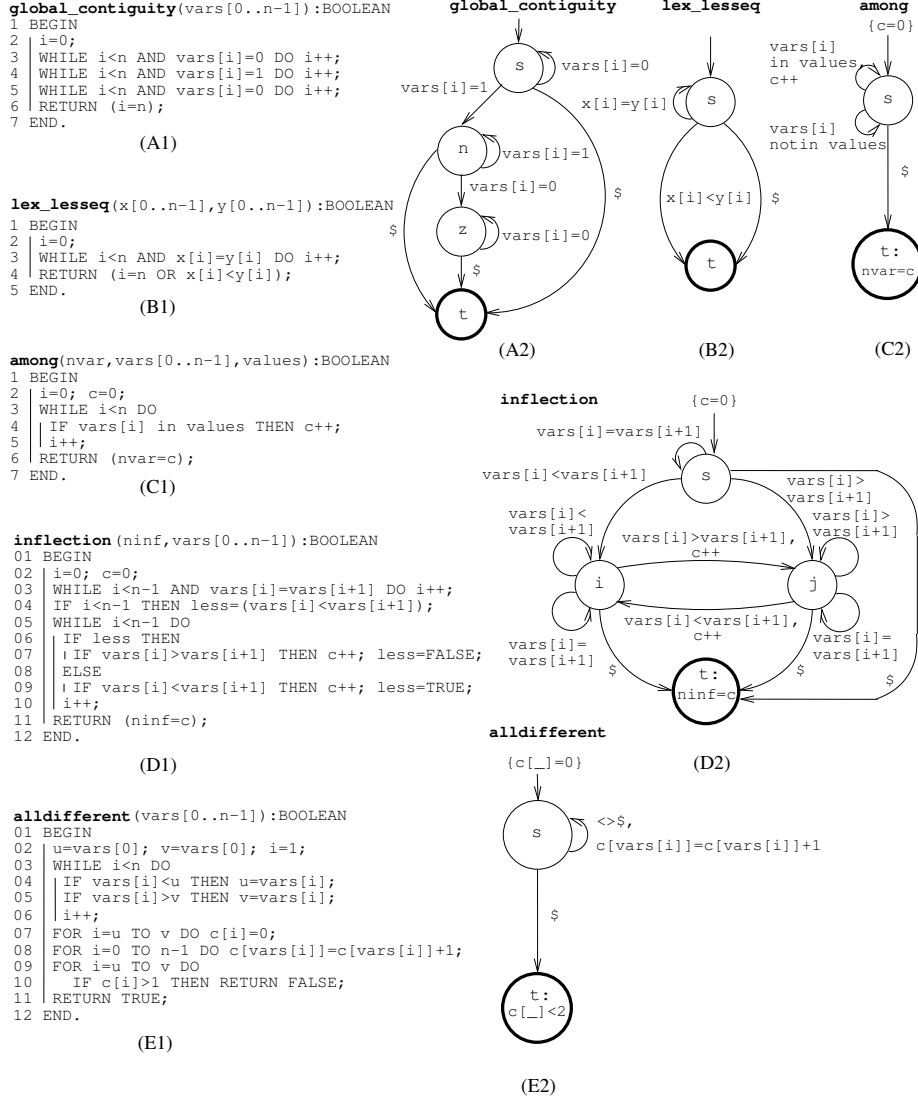


Figure 1.7: Five checkers and their corresponding automata

Parts (A1), (B1), (C1), (D1) and (E1) of Figure 1.7 depict the five checkers respectively associated with `global_contiguity`, with `lex_lesseq`, with `among`, with

`inflexion` and with `alldifferent`. For each checker we observe the following facts:

- Within the checker depicted by part (A1) of Figure 1.7, the values of the sequence $\text{vars}[0], \dots, \text{vars}[n-1]$ are successively compared against 0 and 1 in order to check that we have at most one group of consecutive 1. This can be translated to the automaton depicted by part (A2) of Figure 1.7. The automaton takes as input the sequence $\text{vars}[0], \dots, \text{vars}[n-1]$, and triggers successively a transition for each term of this sequence. Transitions labeled by 0, 1 and \$ are respectively associated with the conditions $\text{vars}[i] = 0$, $\text{vars}[i] = 1$ and $i = n$. Transitions leading to failure are systematically skipped. This is why no transition labeled with a 1 starts from state z .
- Within the checker given by part (B1) of Figure 1.7, the components of vectors \vec{x} and \vec{y} are scanned in parallel. We first skip all the components that are equal and then perform a final check. This is represented by the automaton depicted by part (B2) of Figure 1.7. The automaton takes as input the sequence $\langle x[0], y[0] \rangle, \dots, \langle x[n-1], y[n-1] \rangle$ and triggers a transition for each term of this sequence. Unlike the `global_contiguity` constraint, some transitions now correspond to a condition (e.g. $x[i] = y[i]$, $x[i] < y[i]$) between two variables of the `lex_lesseq` constraint.
- Note that the `among(nvar, vars, values)` constraint involves a variable `nvar` whose value is computed from a given collection of variables `vars`. The checker depicted by part (C1) of Figure 1.7 counts the number of variables of $\text{vars}[0], \dots, \text{vars}[n-1]$ that take their value in `values`. For this purpose it uses a counter c , which is eventually tested against the value of `nvar`. This convinced us to allow the use of counters in an automaton. Each counter has an initial value, which can be updated while triggering certain transitions. The final state of an automaton can force a variable of the constraint to be equal to a given counter. Part (C2) of Figure 1.7 describes the automaton corresponding to the code given in part (C1) of the same figure. The automaton uses the counter variable c initially set to 0 and takes as input the sequence $\text{vars}[0], \dots, \text{vars}[n-1]$. It triggers a transition for each variable of this sequence and increments c when the corresponding variable takes its value in `values`. The final state returns a success when the value of c is equal to `nvar`. At this point we want to stress the following fact: It would have been possible to use an automaton that avoids the use of counters. However, this automaton would depend on the effective value of the argument `nvar`. In addition, it would require more states than the automaton of part (C2) of Figure 1.7. This is typically a problem if we want to have a fixed number of states in order to save memory as well as time.
- As the `among` constraint, the `inflexion(ninf, vars)` constraint involves a variable `ninf` whose value is computed from a given sequence of variables $\text{vars}[0], \dots, \text{vars}[n-1]$. Therefore, the checker depicted in part (D1) of Figure 1.7 uses also a counter c for counting the number of inflexions, and compares its final value to the `ninf` argument. The automaton depicted by part (D2) of Figure 1.7 represents this program. It takes as input the sequence of pairs

$\langle \text{vars}[0], \text{vars}[1] \rangle, \langle \text{vars}[1], \text{vars}[2] \rangle, \dots, \langle \text{vars}[n-2], \text{vars}[n-1] \rangle$ and triggers a transition for each pair. Note that a given variable may occur in more than one pair. Each transition compares the respective values of two consecutive variables of $\text{vars}[0..n-1]$ and increments the counter c when a new inflexion is detected. The final state returns a success when the value of c is equal to minf .

- The checker associated with `alldifferent` is depicted by part (E1) of Figure 1.7. It first initializes an array of counters to 0. The entries of the array correspond to the potential values of the sequence $\text{vars}[0], \dots, \text{vars}[n-1]$. In a second phase the checker computes for each potential value its number of occurrences in the sequence $\text{vars}[0], \dots, \text{vars}[n-1]$. This is done by scanning this sequence. Finally in a third phase the checker verifies that no value is used more than once. These three phases are represented by the automaton depicted by part (E2) of Figure 1.7. The automaton depicted by part (E2) takes as input the sequence $\text{vars}[0], \dots, \text{vars}[n-1]$. Its initial state initializes an array of counters to 0. Then it triggers successively a transition for each element $\text{vars}[i]$ of the input sequence and increments by 1 the entry corresponding to $\text{vars}[i]$. The final state checks that all entries of the array of counters are strictly less than 2, which means that no value occurs more than once in the sequence $\text{vars}[0], \dots, \text{vars}[n-1]$.

Synthesizing all the observations we got from these examples leads to the following remarks and definitions for a given global constraint \mathcal{C} :

- For a given state, no transition can be triggered indicates that the constraint \mathcal{C} does not hold.
- Since all transitions starting from a given state are mutually incompatible all automata are deterministic. Let \mathcal{M} denote the set of mutually incompatible conditions associated with the different transitions of an automaton.
- Let $\mathcal{S}_0, \dots, \mathcal{S}_{m-1}$ denote the sequence of subsets of variables of \mathcal{C} on which the transitions are successively triggered. All these subsets contain the same number of elements and refer to some variables of \mathcal{C} . Since these subsets typically depend on the constraint, we leave the computation of $\mathcal{S}_0, \dots, \mathcal{S}_{m-1}$ outside the automaton. To each subset \mathcal{S}_i of this sequence corresponds a variable S_i with an initial domain ranging over $[\text{min}, \text{min} + |\mathcal{M}| - 1]$, where min is a fixed integer. To each integer of this range corresponds one of the mutually incompatible conditions of \mathcal{M} . The sequences $\mathcal{S}_0, \dots, \mathcal{S}_{m-1}$ and S_0, \dots, S_{m-1} are respectively called the *signature* and the *signature argument* of the constraint. The constraint between S_i and the variables of \mathcal{S}_i is called the *signature constraint* and is denoted by $\Psi_{\mathcal{C}}(S_i, \mathcal{S}_i)$.
- From a pragmatic point of view, the task of writing a constraint checker is naturally done by writing down an imperative program where local variables, arrays, assignment statements and control structures are used. This suggested us to consider deterministic finite automata augmented with local variables and assignment statements on these variables. Regarding control structures, we did not

introduce any extra feature since the deterministic choice of which transition to trigger next seemed to be good enough.

- Many global constraints involve a variable whose value is computed from a given collection of variables. This convinced us to allow the final state of an automaton to optionally return a result. In practice, this result corresponds to the value of a local variable of the automaton in the final state.

1.3.2 Defining an automaton

An automaton \mathcal{A} of a global constraint \mathcal{C} is defined by

$$\langle \textit{Signature}, \textit{SignatureDomain}, \textit{SignatureArg}, \textit{SignatureArgPattern}, \\ \textit{Counters}, \textit{Arrays}, \textit{States}, \textit{Transitions} \rangle$$

where:

- *Signature* is the sequence of variables S_0, \dots, S_{m-1} corresponding to the signature of the constraint \mathcal{C} .
- *SignatureDomain* is an interval that defines the range of possible values of the variables of *Signature*.
- *SignatureArg* is the signature argument $\mathcal{S}_0, \dots, \mathcal{S}_{m-1}$ of the constraint \mathcal{C} . The link between the variables of \mathcal{S}_i and the variable S_i ($0 \leq i < m$) is done by writing down the signature constraint $\Psi_{\mathcal{C}}(S_i, \mathcal{S}_i)$.
- When used, *SignatureArgPattern* defines a symbolic name for each term of *SignatureArg*. These names can be used within the description of a transition for expressing an additional condition for triggering the corresponding transition.
- *Counters* is the, possibly empty, list of all counters used in the automaton \mathcal{A} . Each counter is described by a term $t(\textit{Counter}, \textit{InitialValue}, \textit{FinalVariable})$ where *Counter* is a symbolic name representing the counter, *InitialValue* is an integer giving the value of the counter in the initial state of \mathcal{A} , and *FinalVariable* gives the variable that should be unified with the value of the counter in the final state of \mathcal{A} .
- *Arrays* is the, possibly empty, list of all arrays used in the automaton \mathcal{A} . Each array is described by a term $t(\textit{Array}, \textit{InitialValue}, \textit{FinalConstraint})$ where *Array* is a symbolic name representing the array, *InitialValue* is an integer giving the value of all the entries of the array in the initial state of \mathcal{A} . *FinalConstraint* denotes an existing constraint of the catalog that should hold in the final state of \mathcal{A} . Arguments of this constraint correspond to collections of variables that are bound to array of counters, or to variables that are bound to counters declared in *Counters*. For an array of counters we only consider those entries that are located between the first and the last entries that were modified while triggering a transition of \mathcal{A} .

- *States* is the list of states of \mathcal{A} , where each state has the form $\text{source}(id)$, $\text{sink}(id)$ or $\text{node}(id)$. id is a unique identifier associated with each state. Finally, $\text{source}(id)$ and $\text{sink}(id)$ respectively denote the initial and the final state of \mathcal{A} .
- *Transitions* is the list of transitions of \mathcal{A} . Each transition t has the form $\text{arc}(id_1, \text{label}, id_2)$ or $\text{arc}(id_1, \text{label}, id_2, \text{counters})$. id_1 and id_2 respectively correspond to the state just before and just after t , while label denotes the value that the signature variable should have in order to trigger t . When used, counters gives for each counter of Counters its value after firing the corresponding transition. This value is specified by an arithmetic expression involving counters, constants, as well as usual arithmetic functions such as $+$, $-$, min or max . The order used in the counters list is identical to the order used in Counters .

EXAMPLE: As an illustrative example we give the description of the automaton associated with the $\text{inflexion}(\text{ninf}, \text{vars})$ constraint. We have:

- $\text{Signature} = S_0, S_1, \dots, S_{n-2}$,
- $\text{SignatureDomain} = 0..2$,
- $\text{SignatureArg} = \langle \text{vars}[0], \text{vars}[1] \rangle, \dots, \langle \text{vars}[n-2], \text{vars}[n-1] \rangle$,
- $\text{SignatureArgPattern}$ is not used,
- $\text{Counters} = \text{t}(c, 0, \text{ninf})$,
- $\text{States} = [\text{source}(s), \text{node}(i), \text{node}(j), \text{sink}(t)]$,
- $\text{Transitions} = [\text{arc}(s, 1, s), \text{arc}(s, 2, i), \text{arc}(s, 0, j), \text{arc}(s, \$, t), \text{arc}(i, 1, i), \text{arc}(i, 2, i), \text{arc}(i, 0, j, [c + 1]), \text{arc}(i, \$, t), \text{arc}(j, 1, j), \text{arc}(j, 0, j), \text{arc}(j, 2, i, [c + 1]), \text{arc}(j, \$, t)]$.

The signature constraint relating each pair of variables $\langle \text{vars}[i], \text{vars}[i + 1] \rangle$ to the signature variable S_i is defined as follows: $\Psi_{\text{inflexion}}(S_i, \text{vars}[i], \text{vars}[i + 1]) \equiv \text{vars}[i] > \text{vars}[i + 1] \Leftrightarrow S_i = 0 \wedge \text{vars}[i] = \text{vars}[i + 1] \Leftrightarrow S_i = 1 \wedge \text{vars}[i] < \text{vars}[i + 1] \Leftrightarrow S_i = 2$. The sequence of transitions triggered on the ground instance $\text{inflexion}(4, [3, 3, 1, 4, 5, 5, 6, 5, 6, 3])$ is

$$s \xrightarrow[c=0]{\frac{s}{3=3 \Leftrightarrow S_0=1}} s \xrightarrow[c=1]{\frac{3>1 \Leftrightarrow S_1=0}{1<4 \Leftrightarrow S_2=2}} i \xrightarrow[c=1]{\frac{4<5 \Leftrightarrow S_3=2}{5=5 \Leftrightarrow S_4=1}} i \xrightarrow[c=2]{\frac{5<6 \Leftrightarrow S_5=2}{6>5 \Leftrightarrow S_6=0}} j \xrightarrow[c=2]{\frac{5=5 \Leftrightarrow S_7=1}{5<6 \Leftrightarrow S_8=2}} i \xrightarrow[c=3]{\frac{6>3 \Leftrightarrow S_9=0}{5<6 \Leftrightarrow S_8=2}} j \xrightarrow[c=4]{\frac{\$}{\text{ninf}=4}} t$$

Each transition gives the corresponding condition and, possibly, the value of the counter c just after firing that transition.

Chapter 2

Description of the catalog

Contents

2.1 Which global constraints are included?	57
2.2 Which global constraints are missing?	58
2.3 Searching in the catalog	58
2.3.1 How to see if a global constraint is in the catalog?	58
2.3.2 How to search for all global constraints sharing the same structure	59
Searching from a graph property perspective	59
Searching from an automaton perspective	59
2.3.3 Searching all places where a global constraint is referenced	60
2.4 Figures of the catalog	61
2.5 Keywords attached to the global constraints	62

2.1 Which global constraints are included?

The global constraints of this catalog come from the following sources:

- Existing constraint systems like:
 - Alice [2],
 - CHARME in C,
 - CHIP [38] in Prolog, C and C++ <http://www.cosytec.com>
 - CHOCO [39] in Java <http://choco.sourceforge.net/>
 - ECLAIR [40] in Claire,
 - ECLiPSe [41] in Prolog <http://www-icparc.doc.ic.ac.uk/eclipse>
 - FaCile in OCaml <http://www.recherche.enac.fr/opti/facile/>

- IF/PROLOG in Prolog
http://www.ifcomputer.com/IFProlog/Constraints/home_en.html
 - Ilog Solver [42] in C++ and later in Java <http://www.ilog.com>
 - Koalog in Java <http://www.koalog.com/php/index.php>
 - Mozart [43] in Oz <http://www.mozart-oz.org/>
 - SICStus [44] in Prolog <http://www.sics.se/sicstus/>
- Constraint programming papers mostly from conferences like:
 - The Principles and Practice of Constraint Programming (CP)
<http://www.informatik.uni-trier.de/~ley/db/conf/cp/index.html>
 - The International Joint Conference on Artificial Intelligence (IJCAI)
<http://www.informatik.uni-trier.de/~ley/db/conf/ijcai/index.html>
 - The National Conference on Artificial Intelligence (AAAI)
<http://www.informatik.uni-trier.de/~ley/db/conf/aaai/index.html>
 - The International Conference on Logic Programming (ICLP)
<http://www.informatik.uni-trier.de/~ley/db/conf/iclp/index.html>
 - The International Conference of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR)
<http://www.informatik.uni-trier.de/~ley/db/conf/cpaior/>
 - New constraints inspired by variations of existing constraints, practical applications, combinatorial problems, puzzles or discussions with colleagues.

2.2 Which global constraints are missing?

Constraints with too many arguments (like for instance the original `cycle` [45] constraint with 16 arguments), which are in fact a combination of several constraints, were not directly put into the catalog. Constraints that have complex arguments were also omitted. Beside this, the following constraints should be added in some future version of the catalog: `case` [46], `choquet`, `cumulative_trapeze` [47, 48], `inequality_sum` [49], `no_cycle` [50], `range` [51], `regular` [5], `roots` [51], `soft_gcc` [12], `soft_regular` [12]. Finally we only consider a restricted number of constraints involving set variables since this is a relatively new area, which is currently growing rapidly since 2003.

2.3 Searching in the catalog

2.3.1 How to see if a global constraint is in the catalog?

Searching a given global constraint through the catalog can be achieved in the following ways:

- If you have an idea of the name of the global constraint you are looking for, then put all its letters in lower case, separate distinct words by an underscore and search the resulting name in the index. The entry where the constraint is defined is shown in bold. Common abbreviations or synonyms found in papers have also been put in the index.
- You can also search a global constraint through the list of keywords that is attached to each global constraint. All available keywords are listed alphabetically in Section 2.5 page 62. For each keyword we give the list of global constraints using the corresponding keyword as well as the definition of the keyword.

2.3.2 How to search for all global constraints sharing the same structure

Since we have two ways of defining global constraints (e.g. searching for a graph with specific properties or coming up with an automaton that only recognizes the solutions associated with the global constraint) we can look to the global constraints from these two perspectives.

Searching from a graph property perspective

The index contains all the arc generators as well as all the graph properties and the pages where they are mentioned¹. This allows for finding all global constraints that use a given arc generator or a given graph property in their definition. You can further restrict your search to those global constraints using a specific combination of arc generators and graph properties. All these combinations are listed at the "signature" entry of the index. Within these combinations, a graph property with an underline means that the constraint should be evaluated each time the minimum of this graph property increases. Similarly a graph property with an overline indicates that the constraint should be evaluated each time the maximum of this graph property decreases. For instance if we look for those constraints that both use the *CLIQUE* arc generator as well as the **NARC** graph-property we find the inverse and place_in_pyramid constraints. Since **NARC** is underlined and overlined these constraints will have to be woken each time the minimum or the maximum of **NARC** changes. The signature associated with a global constraint is also shown in the header of the even pages corresponding to the description of the global constraint.

Searching from an automaton perspective

We have created the following list of keywords, which allow for finding all global constraints defined by a specific type of automaton that recognizes its solutions²:

- "automaton" indicates that the catalog provides a deterministic automaton,

¹Arc generators and graph properties are introduced in the section "Describing Explicitly Global Constraints".

²Automata that recognize the solutions of a global constraint were introduced in the section "Describing Explicitly Global Constraints".

- "automaton without counters" indicates that the catalog provides a deterministic automaton without counters as well as without array of counters,
- "automaton with counters" indicates that the catalog provides a deterministic automaton with counters but without array of counters,
- "automaton with array of counters" indicates that the catalog provides a deterministic automaton with array of counters and possibly with counters.

In addition we also provide a list of keywords that characterize the structure of the hypergraph associated with the decomposition of the automaton of a global constraints. Note that, when a global constraint is defined by several graph properties it is also defined by several automata (usually one automata for each graph property). This is for instance the case of the `change_continuity` constraint. Currently we have these keywords:

- "Berge-acyclic constraint network",
- "alpha-acyclic constraint network(2)",
- "alpha-acyclic constraint network(3)",
- "alpha-acyclic constraint network(4)",
- "sliding cyclic(1) constraint network(1)",
- "sliding cyclic(1) constraint network(2)",
- "sliding cyclic(1) constraint network(3)",
- "sliding cyclic(2) constraint network(2)",
- "circular sliding cyclic(1) constraint network(2)",
- "centered cyclic(1) constraint network(1)",
- "centered cyclic(2) constraint network(1)",
- "centered cyclic(3) constraint network(1)".

When a global constraint is only defined by one or several automaton its signature is set to the keyword `AUTOMATON`.

2.3.3 Searching all places where a global constraint is referenced

Beside the page where a global constraint is defined (in bold), the index also gives all the pages where a global constraint is referenced. Since a global constraint can also be used for defining another global constraint the item **Used in** of the description of a global constraint provides this information.

2.4 Figures of the catalog

The catalog contains the following types of figures:

- Figures that illustrate a global constraint or a keyword,
- Figures that depict the initial as well as the final graphs associated with a global constraint,
- Figures that provide an automaton that only recognizes the solutions associated with a given global constraint,
- Figures that give the hypergraph associated with the decomposition of an automaton in terms of signature and transition constraints.

Most of the graph figures that depict the initial and final graph of a global constraint of this catalog were automatically generated by using the open source graph drawing software Graphviz available from AT&T³.

³<http://www.research.att.com/sw/tools/graphviz>

2.5 Keywords attached to the global constraints

This section explains the meaning of the keywords attached to the global constraints of the catalog. For each keyword it first gives the list of global constraints using the corresponding keyword and then defines the keyword. At present the following keywords are in use.

Acyclic:

- `alldifferent_on_intersection`,
- `among_low_up`,
- `arith_or`,
- `cardinality_atleast`,
- `cardinality_atmost`,
- `cardinality_atmost_partition`,
- `change`,
- `change_continuity`,
- `change_pair`,
- `change_partition`,
- `common`,
- `common_interval`,
- `common_modulo`,
- `common_partition`,
- `correspondence`,
- `counts`,
- `cyclic_change`,
- `cyclic_change_joker`.

Denotes the fact that a constraint is defined by one single graph constraint for which the final graph doesn't have any circuit.

All different:

- `alldifferent`,
- `alldifferent_between_sets`,
- `alldifferent_except_0`,
- `alldifferent_interval`,
- `alldifferent_modulo`,
- `alldifferent_on_intersection`,
- `alldifferent_partition`,
- `soft_alldifferent_ctr`,
- `soft_alldifferent_var`,
- `symmetric_alldifferent`,
- `weighted_partial_alldiff`.

Denotes the fact that we have a clique of disequalities or that a constraint is a variation of the `alldifferent` constraint. Variations may be related to relaxations (e.g. `alldifferent_except_0`, `soft_alldifferent_ctr`, `soft_alldifferent_var`), or to specializations (e.g. `symmetric_alldifferent`), of the `alldifferent` constraint. Variations may also result from an extension of the notion of disequality (e.g. `alldifferent_interval`, `alldifferent_modulo`, `alldifferent_partition`).

Alignment:

- orchard.

Denotes the fact that a constraint enforces the alignment of different sets of points.

Alpha-acyclic constraint network(2):

- | | |
|-------------------|-------------------------------|
| • among, | • count, |
| • among_diff_0, | • counts, |
| • among_interval, | • differ_from_at_least_k_pos, |
| • among_low_up, | • exactly, |
| • among_modulo, | • group, |
| • atleast, | • group_skip_isolated_item, |
| • atmost, | • sliding_card_skip0. |

Before defining *alpha-acyclic constraint network(2)* we first need to introduce the following notions:

- The *dual graph* of a constraint network \mathcal{N} is defined in the following way: To each constraint of \mathcal{N} corresponds a vertex in the dual graph and if two constraints have a non-empty set S of shared variables, there is an edge labeled S between their corresponding vertices in the dual graph.
- An edge in the dual graph of a constraint network is *redundant* if its variables are shared by every edge along an alternative path between the two end points [52].
- If the subgraph resulting from the removal of the redundant edges of the dual graph is a tree the original constraint network is called α -acyclic [53].

Alpha-acyclic constraint network(2) denotes an α -acyclic constraint network such that for any pair of constraints the two sets of involved variables share at most two variables.

Alpha-acyclic constraint network(3):

- | | |
|-----------------------------|-----------------------------|
| • group, | • ith_pos_different_from_0. |
| • group_skip_isolated_item, | |

Alpha-acyclic constraint network(3) denotes an α -acyclic constraint network (see *alpha-acyclic constraint network(2)*) such that for any pair of constraints the two sets of involved variables share at most three variables.

Alpha-acyclic constraint network(4):

- max_index,
- min_index.

Alpha-acyclic constraint network(4) denotes an α -acyclic constraint network (see *alpha-acyclic constraint network(2)*) such that for any pair of constraints the two sets of involved variables share at most four variables.

Apertition:

- change_continuity.

Denotes the fact that a constraint is defined by two graph constraints having the same initial graph, where each arc of the initial graph belongs to one of the final graph (but not to both).

Arithmetic constraint:

- product_ctr,
- range_ctr,
- sum_ctr,
- sum_set.

An arithmetic constraint involving a sum, a product, or a difference between a maximum and a minimum value. Such constraints were introduced within the catalog since they are required for defining a given global constraint. For instance the `sum_ctr` constraint is used within the definition of the cumulative constraint.

Array constraint:

- elem,
- element,
- element_lesseq,
- element_greatereq,
- element_matrix,
- element_sparse.

A constraint that allows for expressing simple array equations.

Assignment:

- assign_and_counts,
- assign_and_nvalues,
- balance,
- balance_interval,
- balance_modulo,
- balance_partition,
- bin_packing,
- cardinality_atleast,
- cardinality_atmost,
- global_cardinality,
- global_cardinality_low_up,
- global_cardinality_with_costs,
- indexed_sum,
- interval_and_count,
- interval_and_sum,
- max_nvalue,
- min_nvalue,

- `min_size_set_consecutive_var`,
- `minimum_weight_alldifferent`,
- `same_and_global_cardinality`,
- `sum_of_weights_of_distinct_values`,
- `symmetric_cardinality`,
- `symmetric_gcc`,
- `weighted_partial_alldiff`.

A constraint putting a restriction on all items that are assigned to the same equivalence class or on all equivalence classes that are effectively used. Usually an equivalence class corresponds to one single value (e.g. `balance`, `bin_packing`, `global_cardinality`, `sum_of_weights_of_distinct_values`), to an interval of consecutive values (e.g. `balance_interval`, `interval_and_count`, `interval_and_sum`) or to all values that are congruent modulo a given number (e.g. `balance_modulo`). The restriction on all items that are assigned to the same equivalence class can for instance be a constraint on the number of items (e.g. `cardinality_atleast`, `cardinality_atmost`, `global_cardinality`, `global_cardinality_low_up`) or a constraint on the sum of a specific attribute (e.g. `bin_packing`, `interval_and_sum`).

At least:

- `atleast`,
- `cardinality_atleast`.

A constraint enforcing that one or several values occur a minimum number of time within a given collection of domain variables.

At most:

- `atmost`,
- `cardinality_atmost`,
- `cardinality_atmost_partition`.

A constraint enforcing that one or several values occur a maximum number of time within a given collection of domain variables.

Automaton:

- `alldifferent`,
- `alldifferent_except_0`,
- `alldifferent_interval`,
- `alldifferent_modulo`,
- `alldifferent_on_intersection`,
- `alldifferent_same_value`,
- `among`,
- `among_diff_0`,
- `among_interval`,
- `among_low_up`,
- `among_modulo`,
- `arith`,
- `arith_or`,
- `arith_sliding`,
- `assign_and_counts`,
- `atleast`,
- `atmost`,
- `balance`,
- `balance_interval`,
- `balance_modulo`,
- `bin_packing`,
- `cardinality_atleast`,
- `cardinality_atmost`,

- change,
- change_continuity,
- change_pair,
- circular_change,
- count,
- counts,
- cumulative,
- cyclic_change,
- cyclic_change_joker,
- decreasing,
- deepest_valley,
- differ_from_at_least_k_pos,
- disjoint,
- distance_change,
- domain_constraint,
- elem,
- element,
- element_greatereq,
- element_lesseq,
- element_matrix,
- element_sparse,
- exactly,
- global_cardinality,
- global_contiguity,
- group,
- group_skip_isolated_item,
- heighest_peak,
- in,
- in_same_partition,
- increasing,
- inflexion,
- int_value_precede,
- int_value_precede_chain,
- interval_and_count,
- interval_and_sum,
- inverse,
- ith_pos_different_from_0,
- lex_between,
- lex_different,
- lex_greater,
- lex_greatereq,
- lex_less,
- lex_lesseq,
- longest_change,
- max_index,
- max_nvalue,
- maximum,
- min_index,
- min_n,
- min_nvalue,
- minimum,
- minimum_except_0,
- minimum_greater_than,
- next_element,
- no_peak,
- no_valley,
- not_all_equal,
- not_in,
- nvalue,
- peak,
- same,
- sequence_folding,
- sliding_card_skip0,
- smooth,
- stage_element,
- strictly_decreasing,
- strictly_increasing,
- two_orth_are_in_contact,
- two_orth.do_not_overlap,
- used_by,
- valley.

A constraint for which the catalog provides a deterministic automaton for the ground case. This automaton can usually be used for deriving mechanically a filtering algorithm for the general case. We have the following three types of deterministic automata:

- Deterministic automata without counters and without array of counters,

- Deterministic automata with counters but without array of counters,
- Deterministic automata with array of counters and possibly with counters.

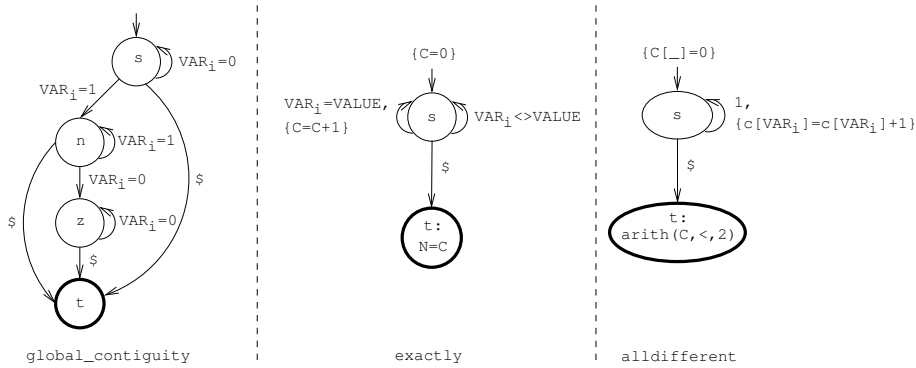


Figure 2.1: Examples of automata

Figure 2.1 shows three automata respectively associated with the `global_contiguity`, the `exactly` and the `alldifferent` constraints. These automata correspond to the three types we described above.

Automaton with array of counters:

- | | |
|---|-------------------------------------|
| • <code>alldifferent</code> , | • <code>cumulative</code> , |
| • <code>alldifferent_except_0</code> , | • <code>disjoint</code> , |
| • <code>alldifferent_interval</code> , | • <code>global_cardinality</code> , |
| • <code>alldifferent_modulo</code> , | • <code>interval_and_count</code> , |
| • <code>alldifferent_on_intersection</code> , | • <code>interval_and_sum</code> , |
| • <code>alldifferent_same_value</code> , | • <code>inverse</code> , |
| • <code>assign_and_counts</code> , | • <code>max_nvalue</code> , |
| • <code>balance</code> , | • <code>min_n</code> , |
| • <code>balance_interval</code> , | • <code>min_nvalue</code> , |
| • <code>balance_modulo</code> , | • <code>nvalue</code> , |
| • <code>bin_packing</code> , | • <code>same</code> , |
| • <code>cardinality_atleast</code> , | • <code>used_by</code> . |
| • <code>cardinality_atmost</code> , | |

A constraint for which the catalog provides a deterministic automaton with array of counters and possibly with counters.

Automaton with counters:

- among,
- among-diff_0,
- among-interval,
- among-low-up,
- among-modulo,
- arith-sliding,
- atleast,
- atmost,
- change,
- change-continuity,
- change-pair,
- circular-change,
- count,
- counts,
- cyclic-change,
- cyclic-change_joker,
- deepest_valley,
- differ_from_at_least_k_pos,
- distance-change,
- exactly,
- group,
- group-skip-isolated-item,
- highest_peak,
- inflexion,
- ith_pos_different_from_0,
- longest-change,
- max-index,
- min-index,
- peak,
- sliding_card_skip0,
- smooth,
- valley.

A constraint for which the catalog provides a deterministic automaton with counters but without array of counters.

Automaton without counters:

- arith,
- arith-or,
- decreasing,
- domain-constraint,
- elem,
- element,
- element-greatereq,
- element-lesseq,
- element-matrix,
- element-sparse,
- global-contiguity,
- in,
- in-same-partition,
- increasing,
- int-value-precede,
- int-value-precede-chain,
- lex-between,
- lex-different,
- lex-greater,
- lex-greatereq,
- lex-less,
- lex-lesseq,
- maximum,
- minimum,
- minimum-except_0,
- minimum-greater-than,
- next-element,
- no-peak,
- no-valley,
- not-all-equal,
- not-in,
- sequence-folding,
- stage-element,
- strictly-decreasing,
- strictly-increasing,
- two-orth-are-in-contact,
- two-orth-do-not-overlap.

A constraint for which the catalog provides a deterministic automaton without counters and without array of counters.

Balanced tree:

- `tree_range`.

A constraint that allows for expressing the fact that we want to cover a digraph by one (or more) *balanced tree*. A *balanced tree* is a tree where no leaf is much farther away than a given threshold from the root than any other leaf. The distance between a leaf and the root of a tree is the number of vertices on the path from the root to the leaf.

Balanced assignment:

- `balance`,
- `balance_interval`,
- `balance_modulo`,
- `balance_partition`.

A constraint that allows for expressing a restriction on the maximum value of the difference between the maximum number of items assigned to the same equivalence class and the minimum number of items assigned to the same equivalence class.

Berge-acyclic constraint network:

- `int_value_precede`,
- `int_value_precede_chain`,
- `global_contiguity`,
- `lex_between`,
- `lex_different`,
- `lex_greater`,
- `lex_greatereq`,
- `lex_less`,
- `lex_lesseq`,
- `two_orth_are_in_contact`,
- `two_orth_do_not_overlap`.

A constraint for which the decomposition associated with its counter-free automaton is *Berge-acyclic*. Arc-consistency for a *Berge-acyclic* constraint network is achieved by making each constraint of the corresponding network arc-consistent. A constraint network for which the corresponding *intersection graph* does not contain any cycle and such that for any pair of constraints the two sets of involved variables share at most one variable is so-called *Berge-acyclic*. The *intersection graph* of a constraint network is built in the following way: to each vertex corresponds a constraint and there is an edge between two vertices if and only if the sets of variables involved in the two corresponding constraints intersect.

Parts (A), (B) and (C) of Figure 2.2 provide three examples of constraint networks, while parts (D), (E) and (F) give their corresponding intersection graph. The constraint network corresponding to part (A) is Berge-acyclic, while the constraint network associated with (B) is not (since its corresponding intersection graph (E) contains a cycle). Finally the constraint network depicted by (C) is also not Berge-acyclic since its third and fourth constraints share more than one variable.

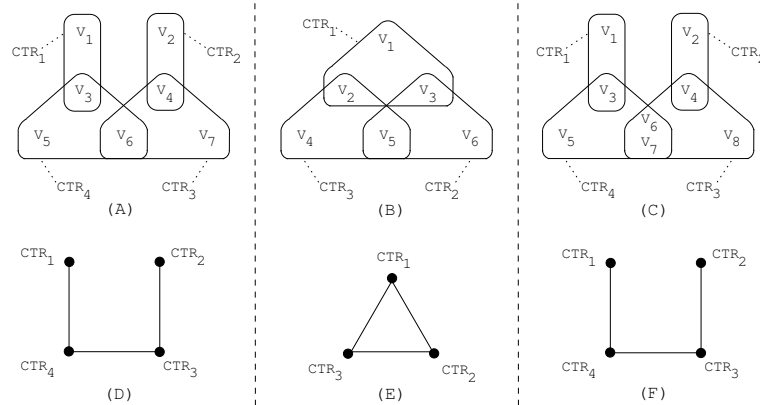


Figure 2.2: Illustration of Berge-acyclic constraint network

Binary constraint:

- `element_greatereq`,
- `element_lesseq`,
- `element_sparse`,
- `eq_set`,
- `stage_element`,
- `sum_set`.

A constraint involving only two variables.

Bioinformatics:

- `all_differ_from_at_least_k_pos`,
- `one_tree`,
- `sequence_folding`.

Denotes the fact that, for a given constraint, either there is a reference to its uses in Bioinformatics, or it was inspired by a problem from the area of Bioinformatics.

Bipartite:

- `alldifferent_on_intersection`,
- `among_low_up`,
- `arith_or`,
- `cardinality_atleast`,
- `cardinality_atmost`,
- `cardinality_atmost_partition`,
- `common`,
- `common_interval`,
- `common_modulo`,
- `common_partition`,
- `correspondence`,
- `counts`.

Denotes the fact that a constraint is defined by one graph constraint for which the final graph is bipartite.

Bipartite matching:

- alldifferent,
- alldifferent_between_sets,
- disjoint,
- lex_alldifferent.

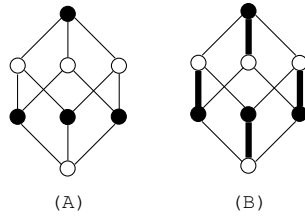


Figure 2.3: A bipartite graph and one of its bipartite matching

Denotes the fact that, for a given constraint, a *bipartite matching* algorithm can be used within its filtering algorithm. A *bipartite matching* is a subgraph that pairs every vertex of a *bipartite graph* with exactly one other vertex. A *bipartite graph* is a graph for which the set of vertices can be partitioned in two parts such that no two vertices in the same part are joined by an edge. Part (A) of Figure 2.3 shows a bipartite graph with a possible division of the vertices in black and white, while part (B) depicts with a thick line a bipartite matching of this graph.

Boolean channel:

- domain_constraint.

A constraint that allows for making the link between a set of 0-1 variables B_1, B_2, \dots, B_n and a domain variable V . It enforces a condition of the form $V = i \Leftrightarrow B_i = 1$.

Border:

- period.

A constraint that can be related to the notion of *border*, which we define now. Given a sequence $s = urv$, r is a *prefix* of s when u is empty, r is a *suffix* of s when v is empty, r is a *proper factor* of s when $r \neq s$. A *border* of a non-empty sequence s is a *proper factor* of s , which is both a *prefix* and a *suffix* of s . We have that the smallest period of a sequence s is equal to the size of s minus the length of the longest border of s .

Bound-consistency:

- alldifferent,
- global_cardinality,
- same,
- used_by.

Denotes the fact that, for a given constraint, there is a filtering algorithm that ensures *bound-consistency* for its variables. A filtering algorithm ensures *bound-consistency* for a given constraint *ctr* if and only if for every variable *V* of *ctr*:

- There exists at least one solution for *ctr* such that $V = \min(V)$ and every other variable *W* of *ctr* is assigned to a value located in its range $\min(W).. \max(W)$,
- There exists at least one solution for *ctr* such that $V = \max(V)$ and every other variable *W* of *ctr* is assigned to a value located in its range $\min(W).. \max(W)$.

One interest of this definition is that it sometimes gives the opportunity to come up with a filtering algorithm that has a lower complexity than the algorithm that achieves arc-consistency. Discarding holes from the variables usually leads to graphs with a specific structure for which one can take advantage in order to derive more efficient graph algorithms. Filtering algorithms that achieve bound-consistency can also be used in a preprocessing phase before applying a more costly filtering algorithm that achieves arc-consistency. Note that there is a second definition of *bound-consistency* where the range $\min(W).. \max(W)$ is replaced by the domain of the variable *W*. However within the context of global constraints all current filtering algorithms don't refer to this second definition.

Centered cyclic(1) constraint network(1):

- domain_constraint,
- in,
- maximum,
- minimum,
- minimum_except_0,
- not_in.

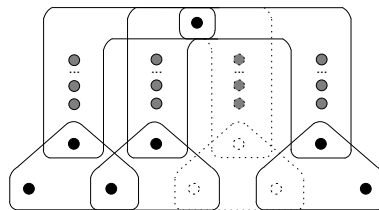


Figure 2.4: Hypergraph associated with a centered cyclic(1) constraint network(1)

A constraint network corresponding to the pattern depicted by Figure 2.4. Circles depict variables, while arcs are represented by a set of variables. Gray circles correspond to optional variables. All pairs of constraints have at most one variable in common.

Centered cyclic(2) constraint network(1):

- `elem,`
- `element,`
- `element_greatereq,`
- `element_lesseq,`
- `element_sparse,`
- `in_same_partition,`
- `minimum_greater_than,`
- `stage_element.`

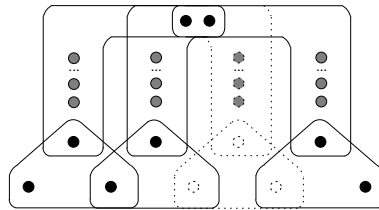


Figure 2.5: Hypergraph associated with a centered cyclic(2) constraint network(1)

A constraint network corresponding to the pattern depicted by Figure 2.5. Circles depict variables, while arcs are represented by a set of variables. Gray circles correspond to optional variables.

Centered cyclic(3) constraint network(1):

- `element_matrix,`
- `next_element.`

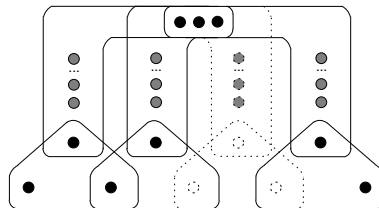


Figure 2.6: Hypergraph associated with a centered cyclic(3) constraint network(1)

A constraint network corresponding to the pattern depicted by Figure 2.6. Circles depict variables, while arcs are represented by a set of variables. Gray circles correspond to optional variables.

Channel routing:

- `connect_points`.

A constraint that can be used for modeling *channel routing* problems. *Channel routing* consists of creating a layout in a rectangular region of a VLSI chip in order to link together the terminals of different modules of the chip. Connections are usually made by wire segments on two different layers: Horizontal wire segments on the first layer are placed along lines called tracks, while vertical wire segments on the second layer connect terminals to the horizontal wire segments, with vias at the intersection.

Channeling constraint:

- `domain_constraint`,
- `inverse`,
- `inverse_set`,
- `link_set_to_booleans`,
- `same`.

Constraints that allow for linking two models of the same problem. Usually channeling constraints show up in the following context:

- When a problem can be modeled by using different types of variables (e.g. 0-1 variables, domain variables, set variables),
- When a problem can be modeled by using two distinct matrices of variables representing the same information redundantly,
- When, in a problem, the roles of the variables and the values can be interchanged. This is typically the case when we have a bijection between a set of variables and the values they can take.

Circuit:

- `circuit`,
- `cutset`,
- `cycle`,
- `symmetric_alldifferent`.

A constraint such that its initial or its final graph corresponds to zero (e.g. `cutset`), one (e.g. `circuit`) or several (e.g. `cycle`, `symmetric_alldifferent`) vertex-disjoint circuits.

Circular sliding cyclic(1) constraint network(2):

- `circular_change`.

A constraint network corresponding to the pattern depicted by Figure 2.7. Circles depict variables, while arcs are represented by a set of variables.

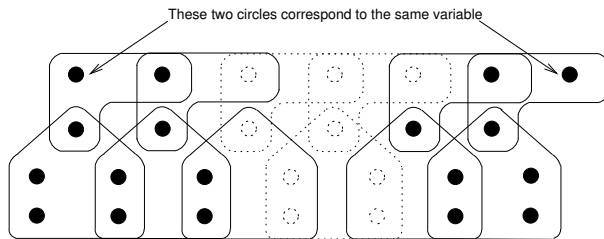


Figure 2.7: Hypergraph corresponding to a circular sliding cyclic(1) constraint network(2)

Cluster:

- `circuit_cluster`.

A constraint that partitions the vertices of an initial graph into several clusters.

Coloured:

- `assign_and_counts`,
- `coloured_cumulative`,
- `coloured_cumulatives`,
- `cycle_card_on_path`,
- `interval_and_count`.

A constraint with a collection where one of the attributes is a color.

Conditional constraint:

- `size_maximal_sequence_alldifferent`,
- `size_maximal_starting_sequence_alldifferent`.

A constraint that allows for expressing the fact that some constraints can be enforced during the enumeration phase.

Connected component:

- `alldifferent_on_intersection`,
- `binary_tree`,
- `change_continuity`,
- `circuit_cluster`,
- `cycle`,
- `cycle_card_on_path`,
- `cycle_resource`,
- `global_contiguity`,
- `group`,
- `k_cut`,
- `map`,
- `nvalue_on_intersection`,
- `temporal_path`,
- `tree`,
- `tree_range`,
- `tree_resource`.

Denotes the fact that a constraint uses in its definition a graph property (e.g. `MAX_NCC`, `MIN_NCC`, `NCC`) constraining the connected components of its associated final graph.

Consecutive loops are connected:

- `group`.

Denotes the fact that the graph constraints of a global constraint use only the *PATH* and the *LOOP* arc generators and that their final graphs do not contain consecutive vertices that have a loop and that are not connected together by an arc.

Consecutive values:

- `max_size_set_of_consecutive_var`,
- `min_size_set_of_consecutive_var`,
- `nset_of_consecutive_values`.

A constraint for which the definition involves the notion of consecutive values assigned to the variables of a collection of domain variables.

Constraint between two collections of variables:

- `common`,
- `common_interval`,
- `common_modulo`,
- `common_partition`,
- `same`,
- `same_and_global_cardinality`,
- `same_intersection`,
- `same_interval`,
- `same_modulo`,
- `same_partition`,
- `soft_same_interval_var`,
- `soft_same_modulo_var`,
- `soft_same_partition_var`,
- `soft_same_var`,
- `soft_used_by_interval_var`,
- `soft_used_by_modulo_var`,
- `soft_used_by_partition_var`,
- `soft_used_by_var`,
- `sort`,
- `used_by`,
- `used_by_interval`,
- `used_by_modulo`,
- `used_by_partition`.

A constraint involving only two collections of domain variables in its arguments.

Constraint between three collections of variables:

- `correspondence`,
- `sort_permutation`.

A constraint involving only three collections of domain variables in its arguments.

Constraint involving set variables:

- `alldifferent_between_sets`,
- `clique`,
- `eq_set`,
- `in_set`,
- `inverse_set`,
- `k_cut`,
- `link_set_to_booleans`,
- `path_from_to`,

- `set_value_precede,`
- `strongly_connected,`
- `sum_set,`
- `symmetric_cardinality,`
- `symmetric_gcc,`
- `tour.`

A constraint involving set variables in its arguments.

Constraint on the intersection:

- `alldifferent_on_intersection,`
- `nvalue_on_intersection,`
- `same_intersection.`

Denotes the fact that a constraint involving two collections of variables imposes a restriction on the values that occur in both collections.

Contact:

- `orths_are_connected,`
- `two_orth_are_in_contact.`

A constraint enforcing that some orthotopes touch each other. Part (A) of Figure 2.8 shows two orthotopes that are in contact while parts (B) and (C) give two examples of orthotopes that are not in contact.

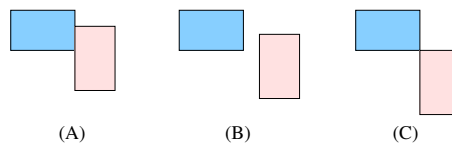


Figure 2.8: Illustration of the notion of contact

Convex:

- `global_contiguity.`

A constraint involving the notion of *convexity*. A subset \mathcal{S} of the plane is called *convex* if and only if for any pair of points p, q of this subset the corresponding line-segment is contained in \mathcal{S} . Part (A) of Figure 2.9 gives an example of convex set, while part (B) depicts an example of non-convex set.

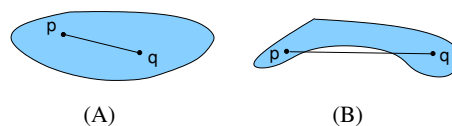


Figure 2.9: A convex set and a non-convex set

Convex hull relaxation:

- `sum`.

Given a non-convex set \mathcal{S} , \mathcal{R} is a *convex outer approximation* of \mathcal{S} if:

- \mathcal{R} is convex,
- If $s \in \mathcal{S}$, then $s \in \mathcal{R}$.

Given a non-convex set \mathcal{S} , \mathcal{R} is the *convex hull* of \mathcal{S} if:

- \mathcal{R} is a convex outer approximation of \mathcal{S} ,
- For every \mathcal{T} where \mathcal{T} is a convex outer approximation of \mathcal{S} , $\mathcal{R} \subseteq \mathcal{T}$.

Part (A) of Figure 2.10 depicts a non-convex set, while part (B) gives its corresponding convex hull.

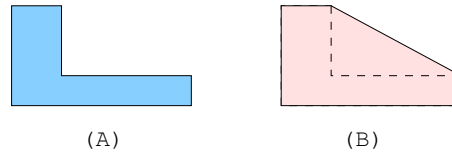


Figure 2.10: Convex hull of a non-convex set

Within the context of linear programming the *convex hull relaxation* of a non-convex set \mathcal{S} corresponds to the set of linear constraints characterizing the convex hull of \mathcal{S} .

Cost filtering constraint:

- `global_cardinality_with_costs`,
- `sum_of_weights_of_distinct_values`,
- `minimum_weight_alldifferent`,
- `weighted_partial_alldiff`.

A constraint that has a set of decision variables as well as a cost variable and for which there exists a filtering algorithm that restricts the state variables from the minimum or maximum value of the cost variable.

Cost matrix:

- `global_cardinality_with_costs`,
- `minimum_weight_alldifferent`.

A constraint for which a first argument corresponds to a collection of variables `Vars`, a second argument to a cost matrix `M`, and a third argument to a cost variable `C`. Let `Vals` denote the set of values that can be assigned to the variables of `Vars`. The cost matrix defines for each pair v, u ($v \in \text{Vars}, u \in \text{Vals}$) an elementary cost, which is used for computing `C` when value u is assigned to variable v .

Counting constraint:

- among,
- among_diff_0,
- among_interval,
- among_low_up,
- among_modulo,
- count,
- counts,
- discrepancy,
- exactly,
- nclass,
- nequivalence,
- ninterval,
- npair,
- nvalue,
- nvalue_on_intersection,
- nvalues,
- nvalues_except_0.

A constraint restricting the number of occurrences of some values (respectively some pairs of values) within a given collection of domain variables (respectively pairs of domain variables).

Cycle:

- cycle,
- symmetric_alldifferent.

A constraint that can be used for restricting the number of cycles of a permutation or for restricting the size of the cycles of a permutation.

Cyclic:

- circular_change,
- cyclic_change,
- cyclic_change_joker,
- stretch_circuit.

A constraint that involves a kind of cyclicity in its definition. It either uses the arc generator *CIRCUIT* or an arc constraint involving `mod`.

Data constraint:

- elem,
- element,
- element_greatereq,
- element_lesseq,
- element_matrix,
- element_sparse,
- elements,
- elements_alldifferent,
- elements_sparse,
- in_relation,
- ith_pos_different_from_0,
- next_element,
- next_greater_element,
- stage_element,
- sum.

A constraint that allows for representing an access to an element of a data structure (e.g. a table, a matrix, a relation) or to compute a value from a given data structure.

Decomposition:

- `all_min_dist`,
- `all_differ_from_at_least_k_pos`,
- `among_seq`,
- `arith`,
- `arith_or`,
- `arith_sliding`,
- `decreasing`,
- `diffn`,
- `diffn_column`,
- `diffn_include`,
- `disjunctive`,
- `domain_constraint`,
- `increasing`,
- `lex_alldifferent`,
- `lex_chain_less`,
- `lex_chain_lesseq`,
- `link_set_to_booleans`,
- `orth_link_ori_siz_end`,
- `sequence_folding`,
- `sliding_distribution`,
- `sliding_sum`,
- `strictly_decreasing`,
- `strictly_increasing`,
- `symmetric_cardinality`,
- `symmetric_gcc`.

A constraint for which the catalog provides a description in terms of a conjunction of more elementary constraints. This is the case when the constraint is described by one or several graph constraints that all satisfy the following property: The description uses the **NARC** graph property and forces all arcs of the initial graph to belong to the final graph. Most of the time we have only one single graph constraint. But some constraints (e.g. `diffn`) use more than one. Note that the arc constraint can sometimes be a logical expression involving several constraints (e.g. `domain_constraint`).

Decomposition-based violation measure:

- `soft_alldifferent_ctr`.

A soft constraint associated to a constraint which can be described in terms of a conjunction of more elementary constraints for which the violation cost is the number of violated elementary constraints.

Demand profile:

- `cumulatives`,
- `same_and_global_cardinality`.

A constraint that allows for representing problems where one has to allocate resources in order to cover a given demand. A profile specifies for each instant the minimum, and possibly maximum, required demand.

Derived collection:

- assign_and_counts,
- correspondence,
- cumulative_two_d,
- cumulative_with_level_of_priority,
- cumulatives,
- cycle_resource,
- domain_constraint,
- element,
- element_matrix,
- element_sparse,
- elements_sparse,
- golomb,
- in,
- in_relation,
- in_same_partition,
- lex_greater,
- lex_greatereq,
- lex_less,
- lex_lesseq,
- link_set_to_booleans,
- minimum_greater_than,
- next_element,
- next_greater_element,
- not_in,
- sliding_time_window_from_start,
- sort_permutation,
- track,
- tree_resource,
- two_layer_edge_crossing.

A constraint that uses one or several derived collections.

Difference:

- golomb.

Denotes the fact that the definition in terms of graph property of a constraint involves a difference between two variables within its arc constraint.

Directed acyclic graph:

- cutset.

A constraint that forces the final graph to be a *directed acyclic graph*. A *directed acyclic graph* is a digraph with no path starting and ending at the same vertex.

Disequality:

- all_differ_from_at_least_k_pos,
- alldifferent,
- alldifferent_between_sets,
- disjoint,
- elements_alldifferent,
- golomb,
- lex_different,
- not_all_equal,
- not_in,
- soft_alldifferent_ctr,
- soft_alldifferent_var,
- symmetric_alldifferent.

Denotes the fact that a disequality between two domain variables, one domain variable and a fixed value, or two set variables is used within the definition of

a constraint. Denotes also the fact that the notion of disequality can be used within the informal definition of a constraint. This is for instance the case for the relaxation of the `alldifferent` constraint (i.e. `soft_alldifferent_ctr`, `soft_alldifferent_var`), which do not strictly enforce a disequality.

Domain channel:

- `domain_constraint`.

A constraint that allows for making the link between a domain variable V and a set of 0-1 variables B_1, B_2, \dots, B_n . It enforces a condition of the form $V = i \Leftrightarrow B_i = 1$.

Domain definition:

- `arith`,
- `in`,
- `not_in`.

A constraint that is used for defining the initial domain of one or several domain variables or for removing some values from the domain of one or several domain variables.

Domination:

- `nvalue`,
- `sum_of_weights_of_distinct_values`.

A constraint that can be used for expressing directly the fact that we search for a *dominating set* in an undirected graph. Given an undirected graph $G = (V, E)$ where V is a finite set of vertices and E a finite set of unordered pairs of distinct elements from V , a set S is a *dominating set* if for every vertex $u \in V - S$ there exists a vertex $v \in S$ such that u is adjacent to v . Part (A) of Figure 2.11 gives an undirected graph G , while part (B) depicts a dominating set $S = \{e, f, g\}$ in G .

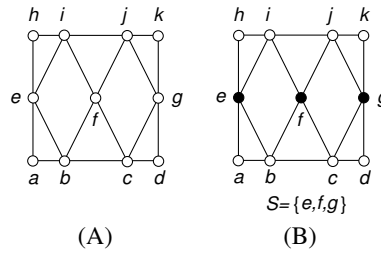


Figure 2.11: A graph and one of its dominating set

Dual model:

- `inverse`,
- `inverse_set`.

A constraint that can be used as a channeling constraint in a problem where the roles of the variables and the values can be interchanged. This is for instance the case when we have a bijection between a set of variables and the values they can take.

Duplicated variables:

- `global_cardinality`,
- `lex_less`,
- `lex_greater`,
- `lex_lesseq`,
- `lex_greatereq`,

A constraint for which the situation where the same variable can occur more than once was considered in order to derive a better filtering algorithm or to prove a complexity result for achieving arc-consistency.

Empty intersection:

- `disjoint`.

A constraint that enforces an empty intersection between two sets of variables.

Equality:

- `eq_set`.

Denotes the fact that the notion of equality can be used within the informal definition of a constraint.

Equality between multisets:

- `same`,
- `same_and_global_cardinality`.

A constraint that can be used for modeling an equality constraint between two multisets.

Equivalence:

- `balance_interval`,
- `nclass`,
- `balance_modulo`,
- `nequivalence`,
- `balance_partition`,
- `ninterval`,
- `balance`,
- `not_all_equal`,
- `max_nvalue`,
- `npair`,
- `min_nvalue`,

- `nvalue,`
- `nvalues,`
- `soft_alldifferent_var.`

Denotes the fact that a constraint is defined by a graph constraint for which the final graph is reflexive, symmetric and transitive.

Euler knight:

- `cycle.`

Denotes the fact that a constraint can be used for modeling the *Euler knight problem*. The *Euler knight problem* consists of finding a sequence of moves on a chessboard by a knight such that each square of the board is visited exactly once.

Excluded:

- `not_in.`

A constraint that prevents certain values to be taken by a variable.

Extension:

- `in_relation.`

A constraint that is defined by explicitly providing all its solutions.

Facilities location problem:

- `cycle_or_accessibility,`
- `sum_of_weights_of_distinct_values.`

A constraint that allows for modeling a facilities location problem. In a facilities location problem one has to select a subset of locations from a given initial set so that a given set of conditions holds.

Flow:

- `global_cardinality,`
- `global_cardinality_low_up,`
- `same,`
- `soft_alldifferent_ctr,`
- `symmetric_cardinality,`
- `symmetric_gcc,`
- `used_by.`

A constraint for which there is a filtering algorithm based on an algorithm that finds a feasible flow in a graph. This graph is constructed from the variables of the constraint as well as from their potential values.

Frequency allocation problem:

- `all_min_dist`.

A constraint that was used for modeling frequency allocation problems.

Functional dependency:

- `elem`,
- `element`,
- `elements`,
- `elements_alldifferent`,
- `stage_element`.

A constraint that allows for representing a *functional dependency* between two domain variables. A variable X is said to *functionally determine* another variable Y if and only if each potential value of X is associated with exactly one potential value of Y .

Geometrical constraint:

- `connect_points`,
- `crossing`,
- `cumulative_two_d`,
- `cycle_or_accessibility`,
- `diffn`,
- `diffn_column`,
- `diffn_include`,
- `graph_crossing`,
- `orchard`,
- `orth_on_the_ground`,
- `orth_on_to_of_orth`,
- `orths_are_connected`,
- `place_in_pyramid`,
- `polyomino`,
- `sequence_folding`,
- `two_layer_edge_crossing`,
- `two_orth_are_in_contact`,
- `two_orth_column`,
- `two_orth_do_not_overlap`,
- `two_orth_include`.

A constraint between geometrical objects (e.g. points, line-segments, rectangles, parallelepipeds, orthotopes) or a constraint selecting a subset of points so that a given geometrical property holds (e.g. distance).

Golomb ruler:

- `golomb`.

A constraint that allows for expressing the *Golomb ruler* problem. A *Golomb ruler* is a set of integers (marks) $a_1 < \dots < a_k$ such that all the differences $a_i - a_j$ ($i > j$) are distinct.

Graph constraint:

- `binary_tree,`
- `circuit,`
- `circuit_cluster,`
- `clique,`
- `cutset,`
- `cycle,`
- `cycle_card_on_path,`
- `cycle_or_accessibility,`
- `cycle_resource,`
- `derangement,`
- `inverse,`
- `k_cut,`
- `map,`
- `one_tree,`
- `path_from_to,`
- `strongly_connected,`
- `symmetric_alldifferent,`
- `temporal_path,`
- `tour,`
- `tree,`
- `tree_range,`
- `tree_resource.`

A constraint that selects a subgraph from a given initial graph so that this subgraph satisfies a given property.

Graph partitioning constraint:

- `binary_tree,`
- `circuit,`
- `cycle,`
- `cycle_resource,`
- `map,`
- `symmetric_alldifferent,`
- `temporal_path,`
- `tree,`
- `tree_range,`
- `tree_resource.`

A constraint that partitions the vertices of a given initial graph and that keeps one single successor for each vertex so that each partition corresponds to a specific pattern.

Guillotine cut:

- `diffn_column,`
- `two_orth_column.`

A constraint that can enforce some kind of *guillotine cut*. In a lot of cutting problems the stock sheet as well as the pieces to be cut are all shaped as rectangles. In a *guillotine cutting pattern* all cuts must go from one edge of the rectangle corresponding to the stock sheet to the opposite edge.

Hall interval:

- alldifferent,
- global_cardinality.

A constraint for which some filtering algorithms take advantage of *Hall intervals*. Given a set of domain variables, a *Hall set* is a set of values $H = \{v_1, v_2, \dots, v_h\}$ such that there are h variables whose domains are contained in H . A *Hall interval* is a Hall set that consists of an interval of values (and can therefore be specified by its endpoints).

Hamiltonian:

- circuit,
- tour.

A constraint enforcing to cover a graph with one Hamiltonian circuit or cycle. This corresponds to finding a circuit (respectively a cycle) passing all the vertices exactly once of a given digraph (respectively undirected graph).

Heuristics:

- discrepancy.

A constraint that was introduced for expressing a heuristics.

Hypergraph:

- among_seq,
- arith_sliding,
- orchard,
- relaxed_sliding_sum,
- size_maximal_sequence_alldifferent,
- size_maximal_starting_sequence_alldifferent,
- sliding_distribution,
- sliding_sum.

Denotes the fact that a constraint uses in its definition at least one arc constraint involving more than two vertices.

Included:

- in,
- in_set.

Enforces that a domain or a set variable take a value within a list of values (possibly one single value).

Inclusion:

- used_by,
- used_by_interval,
- used_by_modulo,
- used_by_partition.

Denotes the fact that a constraint can model the inclusion of one multiset within another multiset. Usually we consider multiset of values (e.g. used_by) but this can also be multisets of equivalence classes (e.g. used_by_interval, used_by_modulo, used_by_partition).

Indistinguishable values:

- int_value_precede,
- int_value_precede_chain,
- set_value_precede.

A constraint which can be used for breaking symmetries of *indistinguishable values*. *Indistinguishable values* in a solution of a problem can be swapped to construct another solution of the same problem.

Interval:

- alldifferent_interval,
- among_interval,
- balance_interval,
- common_interval,
- interval_and_count,
- interval_and_sum,
- ninterval,
- same_interval,
- soft_same_interval_var,
- soft_used_by_interval_var,
- used_by_interval.

Denotes the fact that a constraint puts a restriction related to a set of fixed intervals (or on one fixed interval).

Joker value:

- alldifferent_except_0,
- among_diff_0,
- connect_points,
- cyclic_change_joker,
- ith_pos_different_from_0,
- minimum_except_0,
- nvalues_except_0,
- period_except_0,
- weighted_partial_alldiff.

Denotes the fact that, for some variables of a given constraint, there exist specific values that have a special meaning: for instance they can be assigned without breaking the constraint. As an example consider the `alldifferent_except_0` constraint, which forces a set of variables to take distinct values, except those variables that are assigned to 0.

Lexicographic order:

- allperm,
- lex2,
- lex.between,
- lex_chain_less,
- lex_chain_lesseq,
- lex_greater,
- lex_greatereq,
- lex_less,
- lex_lesseq,
- strict_lex2.

A constraint involving a lexicographic ordering relation in its definition.

Limited discrepancy search:

- discrepancy.

A constraint for simulating limited discrepancy search. *Limited discrepancy search* is useful for problems for which there is a successor ordering heuristics that usually leads directly to a solution. It consists of systematically searching all paths that differ from the heuristic path in at most a very small number of discrepancies.

Linear programming:

- circuit,
- cumulative,
- domain_constraint,
- element_greatereq,
- element_lesseq,
- k_cut,
- link_set_to_booleans,
- path_from_to,
- strongly_connected,
- sum,
- tour.

A constraint for which a reference provides a linear relaxation (e.g. cumulative, sum) or a constraint that was also proposed within the context of linear programming (e.g. circuit, domain_constraint).

Line-segments intersection:

- crossing,
- graph_crossing,
- two_layer_edge_crossing.

A constraint on the number of line-segment intersections.

Magic hexagon:

- `global_cardinality_with_costs`.

A constraint that can be used for modeling the magic hexagon problem. The *magic hexagon* problem consists of finding an arrangement of n hexagons, where an integer from 1 to n is assigned to each hexagon so that:

- Each integer from 1 to n occurs exactly once,
- The sum of the numbers along any straight line is the same.

Figure 2.12 shows a magic hexagon.

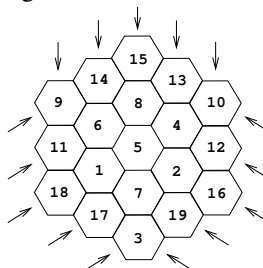


Figure 2.12: A magic hexagon

Magic series:

- `global_cardinality`.

A constraint that allows for modeling the *magic series* problem with one single constraint. A non-empty finite series $S = (s_0, s_1, \dots, s_n)$ is *magic* if and only if there are s_i occurrences of i in S for each integer i ranging from 0 to n . $3, 2, 1, 1, 0, 0, 0$ is an example of such a magic series for $n = 6$.

Magic square:

- `global_cardinality_with_costs`.

A constraint that can be used for modeling the magic square problem. The *magic square* problem consists in filling an n by n square with n^2 distinct integers so that the sum of each row and column and of both main diagonals be the same.

Matching:

- `symmetric_alldifferent`.

A constraint that allows for expressing the fact that we want to find a *perfect matching* on a graph with an even number of vertices. A *perfect matching* on a graph G with n vertices is a set of $n/2$ edges of G such that no two edges have a vertex in common.

Matrix:

- `allperm`,
- `colored_matrix`,
- `element_matrix`,
- `lex2`,
- `strict_lex2`.

A constraint on a matrix of domain variables (e.g. `allperm`, `colored_matrix`, `lex2`, `strict_lex2`) or a constraint that allows for representing the access to an element of a matrix (e.g. `element_matrix`).

Matrix model:

- `allperm`,
- `colored_matrix`,
- `lex2`,
- `strict_lex2`.

A constraint on a matrix of domain variables. A *matrix model* is a model involving one matrix of domain variables.

Matrix symmetry:

- `lex2`,
- `lex_chain_less`,
- `lex_chain_lesseq`,
- `lex_greater`,
- `lex_greatereq`,
- `lex_less`,
- `lex_lesseq`.

A constraint that can be used for breaking certain types of symmetries within a matrix of domain variables.

Maximum:

- `max_index`,
- `max_n`,
- `max_nvalue`,
- `max_size_set_of_consecutive_var`,
- `maximum`,
- `maximum_modulo`.

A constraint for which the definition involves the notion of maximum.

Maximum clique:

- `clique`.

A constraint that can be used for searching for a *maximum clique* in a graph. A *maximum clique* is a clique of maximum size, a clique being a subset of vertices such that each vertex is connected to all other vertices of the clique.

Maximum number of occurrences:

- `max_nvalue`.

A constraint that restricts the maximum number of times that a given value is taken.

maxint:

- `deepest_valley`,
- `min_n`,
- `minimum`,
- `minimum_except_0`,
- `minimum_modulo`.

A constraint that uses `maxint` in its definition in terms of graph properties or in terms of automata. `maxint` is the largest integer that can be represented on a machine.

Minimum:

- `min_index`,
- `min_n`,
- `min_nvalue`,
- `min_size_set_of_consecutive_var`,
- `minimum`,
- `minimum_except_0`,
- `minimum_greater_than`,
- `minimum_modulo`,
- `next_element`,
- `next_greater_element`.

A constraint for which the definition involves the notion of minimum.

Minimum number of occurrences:

- `min_nvalue`.

A constraint that restricts the minimum number of times that a given value is taken.

Modulo:

- `alldifferent_modulo`,
- `among_modulo`,
- `balance_modulo`,
- `common_modulo`,
- `maximum_modulo`,
- `minimum_modulo`,
- `same_modulo`,
- `soft_same_modulo_var`,
- `soft_used_by_modulo_var`,
- `used_by_modulo`.

Denotes the fact that the arc constraint associated with a given constraint mentions the function `mod`.

Multiset:

- `same`,
- `same_and_global_cardinality`.

A constraint using domain variables that can be used for modeling some constraint between multisets.

Multiset ordering:

- `lex_greater`,
- `lex_less`,
- `lex_greatereq`,
- `lex_lesseq`.

Similar constraints exist also within the context of multisets.

no_loop:

- `alldifferent_on_intersection`,
- `change`,
- `all_differ_from_at_least_k_pos`,
- `common_interval`,
- `among_low_up`,
- `common_modulo`,
- `arith_or`,
- `common_partition`,
- `cardinality_atleast`,
- `common`,
- `cardinality_atmost_partition`,
- `correspondence`,
- `cardinality_atmost`,
- `counts`,
- `change_continuity`,
- `crossing`,
- `change_pair`,
- `cyclic_change_joker`,
- `change_partition`,
- `cyclic_change`.

Denotes a constraint defined by a graph constraint for which the final graph doesn't have any loop.

n-queen:

- `alldifferent`,
- `inverse`.

A constraint that can be used for modeling the n-queen problem. Place n queens on a n by n chessboard in such a way that no queen attacks another. Two queens attack each other if they are located on the same column, on the same row or on the same diagonal.

Non-overlapping:

- `diffn`,
- `disjoint_tasks`,
- `orth_on_top_of_orth`,
- `orths_are_connected`,
- `place_in_pyramid`,
- `two_orth_are_in_contact`,
- `two_orth_do_not_overlap`.

A constraint that forces a collection of geometrical objects to not pairwise overlap.

Number of changes:

- `change`,
- `change_pair`,
- `change_partition`,
- `circular_change`,
- `cyclic_change`,
- `cyclic_change_joker`,
- `smooth`.

A constraint restricting the number of times that a given binary constraint holds on consecutive items of a given collection.

Number of distinct equivalence classes:

- `nclass`,
- `nequivalence`,
- `ninterval`,
- `npair`,
- `nvalue`,
- `nvalues`.

A constraint on the number of distinct equivalence classes assigned to a collection of domain variables.

Number of distinct values:

- `assign_and_nvalues`,
- `coloured_cumulative`,
- `coloured_cumulatives`,
- `nvalue`,
- `nvalue_on_intersection`,
- `nvalues`,
- `nvalues_except_0`.

A constraint on the number of distinct values assigned to one or several set of variables.

Obscure:

- `one_tree`.

A constraint for which a better description is needed.

One succ:

- alldifferent_between_sets,
- alldifferent_except_0,
- alldifferent_interval,
- alldifferent_modulo,
- alldifferent_partition,
- alldifferent,
- binary_tree,
- circuit_cluster,
- circuit,
- cycle_card_on_path,
- cycle,
- minimum_weight_alldifferent.

Denotes the fact that a constraint is defined by one single graph constraint such that:

- All the vertices of its initial graph belong to the final graph,
- All the vertices of its final graph have exactly one successor.

Order constraint:

- allperm,
- decreasing,
- increasing,
- int_value_precede,
- int_value_precede_chain,
- lex2,
- lex_between,
- lex_chain_less,
- lex_chain_lesseq,
- lex_greater,
- lex_greatereq,
- lex_less,
- lex_lesseq,
- max_index,
- max_n,
- maximum,
- maximum_modulo,
- min_index,
- min_n,
- minimum,
- minimum_except_0,
- minimum_greater_than,
- minimum_modulo,
- next_greater_element,
- set_value_precede,
- strict_lex2,
- strictly_decreasing,
- strictly_increasing.

A constraint involving an *ordering relation* in its definition. An *ordering relation* R on a set S is a relation such that, for every $a, b, c \in S$:

- $a R b$ or $b R a$,
- If $a R b$ and $b R c$, then $a R c$,
- If $a R b$ and $b R a$ then $a = b$.

Orthotope:

- `diffn`,
- `diffn_column`,
- `diffn_include`,
- `orth_link_ori_siz_end`,
- `orth_on_the_ground`,
- `orth_on_top_of_orth`,
- `orths_are_connected`,
- `place_in_pyramid`,
- `two_orth_are_in_contact`,
- `two_orth_column`,
- `two_orth_do_not_overlap`,
- `two_orth_include`.

A constraint involving *orthotopes*. An *orthotope* corresponds to the generalization of the rectangle and box to the n -dimensional case.

Pair:

- `change_pair`,
- `npair`.

A constraint involving a collection of pairs of variables.

Partition:

- `alldifferent_partition`,
- `balance_partition`,
- `cardinality_atmost_partition`,
- `change_partition`,
- `common_partition`,
- `in_same_partition`,
- `nclass`,
- `same_partition`,
- `soft_same_partition_var`,
- `soft_used_by_partition_var`,
- `used_by_partition`.

A constraint involving in one of its argument a partitioning of a given finite set of integers.

Path:

- `path_from_to`,
- `temporal_path`.

A constraint allowing for expressing the fact that we search for one or several vertex-disjoint *simple paths*. Within a digraph a *simple path* is a set of links that are traversed in the same direction and such that each vertex of the simple path is visited exactly once.

Pentomino:

- `polyomino`.

Can be used to model a *pentomino*. A *pentomino* is an arrangement of five unit squares that are joined along their edges.

Periodic:

- `period`,
- `period_except_0`.

A constraint that can be used for modeling the fact that we are looking for a sequence that has some kind of periodicity.

Permutation:

- `alldifferent`,
- `change_continuity`,
- `circuit`,
- `correspondence`,
- `cycle`,
- `derangement`,
- `elements_alldifferent`,
- `inverse`,
- `same`,
- `same_and_global_cardinality`,
- `same_interval`,
- `same_modulo`,
- `same_partition`,
- `sort`,
- `sort_permutation`,
- `symmetric_alldifferent`.

A constraint that can be used for modeling a permutation or a specific type or characteristic of a permutation. A *permutation* is a rearrangement of elements, where none are changed, added or lost.

Permutation channel:

- `inverse`.

A constraint that allows for modeling the link between a *permutation* and its *inverse permutation*. A *permutation* is a rearrangement of n distinct integers between 1 and n , where none are changed, added or lost. An *inverse permutation* is a permutation in which each number and the number of its position are swapped.

Phylogeny:

- `one_tree`.

A constraint inspired by the area of phylogeny. Phylogeny is concerned by the classification of organism based on genetic connections between species.

Pick-up delivery:

- `cycle`.

A constraint that was used for modeling a *pick-up delivery problem*. In a *pick-up delivery problem*, vehicles have to transport loads from origins to destinations without any transshipment at intermediate locations.

Polygon:

- `diffn`.

A constraint that can be generalized to handle polygons.

Positioning constraint:

- `diffn_column`,
- `diffn_include`,
- `two_orth_column`,
- `two_orth_include`.

A constraint restricting the relative positioning of two or more geometrical objects.

Predefined constraint:

- `allperm`,
- `colored_matrix`,
- `eq_set`,
- `in_set`,
- `lex2`,
- `pattern`,
- `period`,
- `period_except_0`,
- `set_value_precede`,
- `strict_lex2`.

A constraint for which the meaning is not explicitly described in terms of graph properties or in terms of automata.

Producer-consumer:

- `cumulative`,
- `cumulatives`.

A constraint that can be used for modeling problems where a first set of tasks produces a resource, while a second set of tasks consumes this resource. The constraint allows for imposing a limit on the minimum or the maximum stock at each instant.

Product:

- `cumulative_product`,
- `product_ctr`.

A constraint involving a product in its definition.

Proximity constraint:

- `alldifferent_same_value`,
- `distance_change`,
- `distance_between`,

A constraint restricting the distance between two collections of variables according to some measure.

Range:

- `range_ctr`.

An arithmetic constraint involving a difference between a maximum and a minimum value.

Rank:

- `max_n`,
- `min_n`.

A positioning constraint according to an ordering relation.

Relation:

- `in_relation`,
- `symmetric_cardinality`,
- `symmetric_gcc`.

A constraint that allows for representing the access to an element of a *relation* or to model a *relation*. A *relation* is a subset of the product of several finite sets.

Relaxation:

- `alldifferent_except_0`,
- `relaxed_sliding_sum`,
- `soft_alldifferent_ctr`,
- `soft_alldifferent_var`,
- `soft_same_interval_var`,
- `soft_same_modulo_var`,
- `soft_same_partition_var`,
- `soft_same_var`,
- `soft_used_by_interval_var`,
- `soft_used_by_modulo_var`,
- `soft_used_by_partition_var`,
- `soft_used_by_var`,
- `sum_of_weights_of_distinct_values`,
- `weighted_partial_alldiff`.

Denotes the fact that a constraint allows for specifying a partial degree of satisfaction.

Resource constraint:

- `bin_packing`,
- `coloured_cumulative`,
- `coloured_cumulatives`,
- `cumulative`,
- `cumulative_product`,
- `cumulative_with_level_of_priority`,
- `cumulatives`,
- `cycle_resource`,
- `disjunctive`,
- `interval_and_count`,
- `interval_and_sum`,
- `track`,
- `tree_resource`.

A constraint restricting the utilization of a resource. The utilization of a resource is computed from all items that are assigned to that resource.

Run of a permutation:

- `change_continuity`.

A constraint that can be used for putting a restriction on the size of the longest *run* of a permutation. A *run* is a maximal increasing contiguous subsequence in a permutation.

Scalar product:

- `global_cardinality_with_costs`.

A constraint that can be used for modeling a scalar product constraint.

Sequence:

- | | |
|-------------------------------------|--|
| • <code>among_seq</code> , | • <code>period_except_0</code> , |
| • <code>arith_sliding</code> , | • <code>relaxed_sliding_sum</code> , |
| • <code>cycle_card_on_path</code> , | • <code>sequence_folding</code> , |
| • <code>deepest_valley</code> , | • <code>size_maximal_sequence_alldifferent</code> , |
| • <code>highest_peak</code> , | • <code>size_maximal_starting_sequence_alldifferent</code> , |
| • <code>inflexion</code> , | • <code>sliding_card_skip0</code> , |
| • <code>no_peak</code> , | • <code>sliding_distribution</code> , |
| • <code>no_valley</code> , | • <code>sliding_sum</code> , |
| • <code>peak</code> , | • <code>valley</code> . |
| • <code>period</code> , | |

Constrains consecutive variables (possibly not all) of a given collection of domain variables or consecutive vertices of a simple path or a simple circuit. Also a constraint restricting a variable (when fixed to 0 the variable may be omitted) according to consecutive variables of a given collection of domain variables.

Set channel:

- | | |
|------------------------------|---------------------------------------|
| • <code>inverse_set</code> , | • <code>link_set_to_booleans</code> . |
|------------------------------|---------------------------------------|

A channeling constraint involving one or several set variables.

Scheduling constraint:

- | | |
|--|----------------------------------|
| • <code>coloured_cumulative</code> , | • <code>cumulatives</code> , |
| • <code>coloured_cumulatives</code> , | • <code>disjoint_tasks</code> , |
| • <code>cumulative</code> , | • <code>disjunctive</code> , |
| • <code>cumulative_product</code> , | • <code>period</code> , |
| • <code>cumulative_with_level_of_priority</code> , | • <code>period_except_0</code> . |

- `shift`.

A constraint useful for the area of *scheduling*. *Scheduling* is concerned with the allocation or assignment of resources (e.g. manpower, machines, money), over time, to a set of tasks.

Shared table:

- `elements`,
- `elements_sparse`.

A constraint for which the same table is shared by several `element` constraints.

Sliding cyclic(1) constraint network(1):

- `decreasing`,
- `increasing`,
- `no_peak`,
- `no_valley`,
- `not_all_equal`,
- `strictly_decreasing`,
- `strictly_increasing`.

A constraint network corresponding to the pattern depicted by Figure 2.13. Circles depict variables, while arcs are represented by a set of variables.

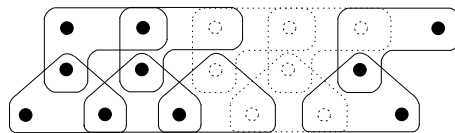


Figure 2.13: Hypergraph associated with a sliding cyclic(1) constraint network(1)

Sliding cyclic(1) constraint network(2):

- `change`,
- `change_continuity`,
- `cyclic_change`,
- `cyclic_change_joker`,
- `deepest_valley`,
- `highest_peak`,
- `inflexion`,
- `peak`,
- `smooth`,
- `valley`.

A constraint network corresponding to the pattern depicted by Figure 2.14. Circles depict variables, while arcs are represented by a set of variables.

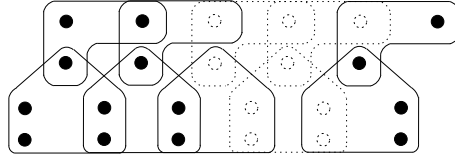


Figure 2.14: Hypergraph associated with a sliding cyclic(1) constraint network(2)

Sliding cyclic(1) constraint network(3):

- change,
- change_continuity,
- longest_change.

A constraint network corresponding to the pattern depicted by Figure 2.15. Circles depict variables, while arcs are represented by a set of variables.

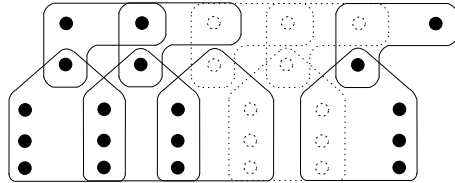


Figure 2.15: Hypergraph associated with a sliding cyclic(1) constraint network(3)

Sliding cyclic(2) constraint network(2):

- change_pair,
- distance_change.

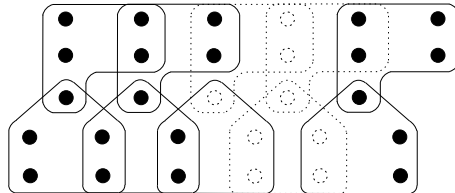


Figure 2.16: Hypergraph associated with a sliding cyclic(2) constraint network(2)

A constraint network corresponding to the pattern depicted by Figure 2.16. Circles depict variables, while arcs are represented by a set of variables.

Sliding sequence constraint:

- `among_seq`,
- `arith_sliding`,
- `cycle_card_on_path`,
- `pattern`,
- `relaxed_sliding_sum`,
- `sliding_card_skip0`,
- `sliding_distribution`,
- `size_maximal_sequence_alldifferent`,
- `size_maximal_starting_sequence_alldifferent`,
- `sliding_sum`,
- `sliding_time_window`,
- `sliding_time_window_from_start`,
- `sliding_time_window_sum`,
- `stretch_circuit`,
- `stretch_path`.

A constraint enforcing a condition on sliding sequences of domain variables that partially overlap or a constraint computing a quantity from a set of sliding sequences. These sliding sequences can be either initially given or dynamically constructed. In the latter case they can for instance correspond to adjacent vertices of a path that has to be built.

Soft constraint:

- | | |
|--|---|
| • <code>relaxed_sliding_sum</code> , | • <code>soft_same_var</code> , |
| • <code>soft_alldifferent_ctr</code> , | • <code>soft_used_by_interval_var</code> , |
| • <code>soft_alldifferent_var</code> , | • <code>soft_used_by_modulo_var</code> , |
| • <code>soft_same_interval_var</code> , | • <code>soft_used_by_partition_var</code> , |
| • <code>soft_same_modulo_var</code> , | • <code>soft_used_by_var</code> , |
| • <code>soft_same_partition_var</code> , | • <code>weighted_partial_alldiff</code> . |

A constraint that is a relaxed form of one other constraint.

Sort:

- `sort`,
- `sort_permutation`.

A constraint involving the notion of sorting in its definition.

Sparse functional dependency:

- `element_sparse`,
- `elements_sparse`.

A constraint that allows for representing a *functional dependency* between two domain variables, where both variables have a restricted number of values. A variable X is said to *functionally determine* another variable Y if and only if each potential value of X is associated with exactly one potential value of Y .

Sparse table:

- `element_sparse`,
- `elements_sparse`.

An `element` constraint for which the table is sparse.

Sport timetabling:

- `symmetric_alldifferent`.

A constraint used for creating sports schedules.

Squared squares:

- `cumulative`,
- `diffn`.

A constraint that can be used for modeling the *squared squares* problem: It consists of tiling a square with smaller squares such that each of the smaller squares has a different integer size.

Strongly connected component:

- `connect_points`,
- `cycle`,
- `cycle_or_accessibility`,
- `cycle_resource`,
- `group_skip_isolated_item`,
- `nclass`,
- `nequivalence`,
- `ninterval`,
- `npair`,
- `nset_of_consecutive_values`,
- `nvalue`,
- `nvalues`,
- `nvalues_except_0`,
- `polyomino`,
- `soft_alldifferent_var`,
- `strongly_connected`.

Denotes the fact that a constraint restricts the strongly connected components of its associated final graph. This is usually done by using a graph property like

MAX_NSCC, MIN_NSCC or NSCC.

Sum:

- sliding_sum,
- sliding_time_window_sum,
- sum,
- sum_ctr,
- sum_set.

A constraint involving one or several sums.

Sweep:

- diffn.

A constraint for which the filtering algorithm may use a *sweep algorithm*. A *sweep algorithm* solves a problem by moving an imaginary object (usually a line or a plane). The object does not move continuously, but only at particular points where we actually do something. A sweep algorithm uses the following two data structures:

- A data structure called the *sweep status*, which contains information related to the current position of the object that moves,
- A data structure named the *event point series*, which holds the events to process.

The algorithm initializes the sweep status for the initial position of the imaginary object. Then the object jumps from one event to the next event; each event is handled by updating the status of the sweep.

Symmetry:

- allperm,
- int_value_precede,
- int_value_precede_chain,
- lex2,
- lex_between,
- lex_chain_less,
- lex_chain_lesseq,
- lex_greater,
- lex_greatereq,
- lex_less,
- lex_lesseq,
- set_value_precede,
- strict_lex2.

A constraint that can be used for breaking certain types of symmetries.

Symmetric:

- connect_points.

Denotes the fact that a constraint is defined by a graph constraint for which the final graph is symmetric.

Table:

- elem,
- element,
- element_greatereq,
- element_lesseq,
- element_sparse,
- elements,
- elements_alldifferent,
- elements_sparse,
- ith_pos_different_from0,
- next_element,
- next_greater_element,
- stage_element.

A constraint that allows for representing the access to an element of a table.

Temporal constraint:

- coloured_cumulative,
- coloured_cumulatives,
- cumulative,
- cumulative_product,
- cumulative_with_level_of_priority,
- cumulatives,
- disjoint_tasks,
- interval_and_count,
- interval_and_sum,
- shift,
- sliding_time_window,
- sliding_time_window_from_start,
- sliding_time_window_sum,
- track.

A constraint involving the notion of time.

Ternary constraint:

- element_matrix.

A constraint involving only three variables.

Timetabling constraint:

- change,
- change_continuity,
- change_pair,
- change_partition,
- circular_change,
- colored_matrix,
- cyclic_change,
- cyclic_change_joker,
- group,
- group_skip_isolated_item,
- interval_and_count,
- interval_and_sum,
- longest_change,
- pattern,
- period,
- period_except0,
- shift,
- sliding_card_skip0,
- smooth,
- stretch_circuit,
- stretch_path,
- symmetric_alldifferent,
- symmetric_cardinality,
- symmetric_gcc,
- track.

Value constraint:

- all_min_dist ,
- alldifferent ,
- alldifferent_except_0 ,
- alldifferent_interval ,
- alldifferent_modulo ,
- alldifferent_on_intersection ,
- alldifferent_partition ,
- among ,
- among_diff_0 ,
- among_interval ,
- among_low_up ,
- among_modulo ,
- arith ,
- arith_or ,
- atleast ,
- atmost ,
- balance ,
- balance_interval ,
- balance_modulo ,
- balance_partition ,
- cardinality_atleast ,
- cardinality_atmost ,
- cardinality_atmost_partition ,
- count ,
- counts ,
- differ_from_at_least_k_pos ,
- discrepancy ,
- disjoint ,
- exactly ,
- global_cardinality ,
- global_cardinality_low_up ,
- in ,
- in_same_partition ,
- in_set ,
- link_set_to_booleans ,
- max_nvalue ,
- max_size_set_of_consecutive_var ,
- min_nvalue ,
- min_size_set_of_consecutive_var ,
- not_all_equal ,
- not_in ,
- nset_of_consecutive_values ,
- same_and_global_cardinality ,
- soft_alldifferent_ctr ,
- soft_alldifferent_var ,
- vec_eq_tuple .

A constraint that puts a restriction on how values can be assigned to usually one or several collections of variables, or possibly one or two variables. These variables usually correspond to domain variables but can sometimes be set variables.

Value partitioning constraint:

- nclass ,
- nequivalence ,
- ninterval ,
- npair ,
- nvalue ,
- nvalues ,
- nvalues_except_0 .

A constraint involving a partitioning of values in its definition.

Value precedence:

- `int_value_precede`,
- `int_value_precede_chain`,
- `set_value_precede`.

A constraint that allows for expressing symmetries between values that are assigned to variables.

Variable-based violation measure:

- `soft_alldifferent_var`,
- `soft_same_interval_var`,
- `soft_same_modulo_var`,
- `soft_same_partition_var`,
- `soft_same_var`,
- `soft_used_by_interval_var`,
- `soft_used_by_modulo_var`,
- `soft_used_by_partition_var`,
- `soft_used_by_var`.

A soft constraint for which the violation cost is the minimum number of variables to unassign in order to get back to a solution.

Variable indexing:

- `indexed_sum`,
- `elem`,
- `element`,
- `element_greatereq`,
- `element_lesseq`,
- `element_sparse`.

A constraint where one or several variables are used as an index into an array.

Variable subscript:

- `indexed_sum`,
- `elem`,
- `element`,
- `element_greatereq`,
- `element_lesseq`.

A constraint that can be used to model one or several variables that have a variable subscript.

Vector:

- `all_differ_from_at_least_k_pos`,
- `differ_from_at_least_k_pos`,
- `lex_alldifferent`,
- `lex_between`,
- `lex_chain_less`,
- `lex_chain_lesseq`,
- `lex_different`,
- `lex_greater`,
- `lex_greatereq`,
- `lex_less`,
- `lex_lesseq`.

Denotes the fact that one (or more) argument of a constraint corresponds to a collection of vectors that all have the same number of components.

Vpartition:

- `group`.

Denotes the fact that a constraint is defined by two graph constraints \mathcal{C}_1 and \mathcal{C}_2 such that:

- The two graph constraints have the same initial graph G_i ,
- Each vertex of the initial graph G_i belongs to exactly one of the final graphs associated with \mathcal{C}_1 and \mathcal{C}_2 .

Weighted assignment:

- `global_cardinality_with_costs`,
- `sum_of_weights_of_distinct_values`,
- `minimum_weight_alldifferent`,
- `weighted_partial_alldiff`.

A constraint expressing an assignment problem such that a cost can be computed from each solution.

Workload covering:

- `cumulatives`.

A constraint that can be used for modeling problems where a first set of tasks \mathcal{T}_1 has to cover a second set of tasks \mathcal{T}_2 . Each task of \mathcal{T}_1 and \mathcal{T}_2 is defined by an origin, a duration and a height. At each point in time t the sum of the heights of the tasks of the first set \mathcal{T}_1 that overlap t has to be greater than or equal to the sum of the heights of the tasks of the second set \mathcal{T}_2 that also overlap t .

Chapter 3

Further topics

Contents

3.1 Differences from the 2000 report	111
3.2 Graph invariants	114
3.2.1 Graph classes	115
3.2.2 Format of an invariant	116
3.2.3 Using the database of invariants	117
3.2.4 The database of graph invariants	118
Graph invariants involving one characteristic of a final graph	121
Graph invariants involving two characteristics of a final graph	123
Graph invariants involving three characteristics of a final graph	131
Graph invariants involving four characteristics of a final graph	144
Graph invariants involving five characteristics of a final graph	149
Graph invariants relating two characteristics of two final graphs	150
Graph invariants relating three characteristics of two final graphs	152
Graph invariants relating four characteristics of two final graphs	153
Graph invariants relating five characteristics of two final graphs	154
Graph invariants relating six characteristics of two final graphs	159
3.3 The electronic version of the catalog	160

3.1 Differences from the 2000 report

This section summarizes the main differences with the SICS report [3] as well as of the corresponding paper [1]. The main differences are listed below:

- We have both simplified and extended the way to generate the vertices of the initial graph and we have introduced a new way of defining set of vertices. We

have also removed the `CLIQUE(MAX)` set of vertices generator since it cannot in general be evaluated in polynomial time. Therefore, we have modified the description of the constraints `assign_and_counts`, `assign_and_nvalues`, `interval_and_count`, `interval_and_sum`, `bin_packing`, `cumulative`, `cumulatives`, `coloured_cumulative`, `coloured_cumulatives`, `cumulative_two_d`, which all used this feature.

- We have introduced the new arc generators `PATH_1` and `PATH_N`, which allow for specifying an n -ary constraint for which n is not fixed. The `size_maximal_starting_sequence_alldifferent` and the `size_maximal_sequence_alldifferent` are examples of global constraints that use these arc generators in order to generate a set of sliding `alldifferent` constraints.
- In addition to traditional domain variables we have introduced `float`, `set` and `multiset` variables as well as several global constraints mentioning float and set variables (see for instance the `choquet` and the `alldifferent_between_sets` constraints). This decision was initially motivated by the fact that several constraint systems and papers mention global constraints dealing with these types of variables. Later on, we realized that set variables also greatly simplify the interface of existing global constraints. This was especially true for those global constraints that explicitly deal with a graph, like `clique` or `cutset`. In this context, using a set variable for catching the successors of a vertex is quite natural. This is especially true when a vertex of the final graph can have more than one successor since it allows for avoiding a lot of 0-1 variables.
- We have introduced the possibility of using more than one graph constraint for defining a given global constraint (see for instance the `cumulative` or the `sort` constraints). Therefore we have removed the notion of dual graph, which was initially introduced in the original report. In this context, we now use two graph constraints (see for instance `change_continuity`).
- On the one hand, we have introduced the following new graph characteristics:

– <code>MAX_DRG</code> ,	– <code>PRODUCT</code> ,
– <code>MAX_OD</code> ,	– <code>RANGE</code> ,
– <code>MIN_DRG</code> ,	– <code>RANGE_DRG</code> ,
– <code>MIN_ID</code> ,	– <code>RANGE_NCC</code> ,
– <code>MIN_OD</code> ,	– <code>SUM</code> ,
– <code>NTREE</code> ,	– <code>SUM_WEIGHT_ARC</code> .
– <code>PATH_FROM_TO</code> ,	

On the other hand, we have removed the following graph characteristics:

- `NCC(COMP, val)`,
- `NSCC(COMP, val)`,

- NTREE(ATTR, COMP, val),
- NSOURCE_EQ_NSINK,
- NSOURCE_GREATEREQ_NSINK.

Finally, MAX_IN_DEGREE has been renamed MAX_ID.

- We have introduced an iterator over the items of a collection in order to specify in a generic way a set of similar elementary constraints or a set of similar graph properties. This was required for describing some global constraints such as `global_cardinality`, `cycle_resource` or `stretch`. All these global constraints mention a condition involving some limit depending on the specific values that are effectively used. For instance the `global_cardinality` constraint forces each value v to be respectively used at least `atleastv` and at most `atmostv` times. This iterator was also necessary in the context of graph covering constraints where one wants to cover a digraph with some patterns. Each pattern consists of one resource and several tasks. One can now attach specific constraints to the different resources. Both the `cycle_resource` and the `tree_resource` constraints illustrate this point.
- We have added some standard existing global constraints that were obviously missing from the previous report. This was for instance the case of the `element` constraint.
- In order to make clear the notion of *family* of global constraints we have computed for each global constraint a *signature*, which summarizes its structure. Each signature was inserted into the index so that one can retrieve all the global constraints sharing the same structure.
- We have generalized some existing global constraints. For instance the `change_pair` constraint extends the `change` constraint. Finally we have introduced some novel global constraints like `disjoint_tasks` or `symmetric_gcc`.
- We have defined the rules for specifying arc constraints.

3.2 Graph invariants

Within the scope of the graph-based description this section shows how to use implied constraints, which are systematically linked to the description of a global constraint. This usually occurs in the following context:

- Quite often, it happens that one wants to enforce the final graph to satisfy more than one graph property. In this context, these graph properties involve several graph characteristics that cannot vary independently.

EXAMPLE: As a practical example, consider the group constraint and its first graph constraint. It involves the four graph characteristics `NCC`, `MIN_NCC`, `MAX_NCC` and `NVERTEX`, which respectively correspond to the number of connected components, the number of vertices of the smallest connected component, the number of vertices of the largest connected component and the number of vertices of the final graph. In this example the number of connected components of the final graph cannot vary independently from the size of the smallest connected component. The same remark applies also for the size of the largest connected component. Having a graph invariant that directly relates the four graph characteristics can dramatically improve the propagation.

- Even if the description of a global constraint involves one single graph characteristic `C`, we can introduce the number of vertices, `NVERTEX`, and the number of arcs, `NARC`, of the final digraph. In this context, we can take advantage of graph invariants linking `C`, `NARC` and `NVERTEX`.
- It also happens that we enforce two graph constraints \mathcal{GC}_1 and \mathcal{GC}_2 that have the same initial graph G . In this context we consider the following situations:
 - Each arc of G belongs to one of the final graphs associated with \mathcal{GC}_1 or with \mathcal{GC}_2 (but not to both). An example of such global constraint is the `change_continuity` constraint. Within the graph invariants this situation is denoted by `apartition`.
 - Each vertex of G belongs to one of the final graphs associated with \mathcal{GC}_1 or with \mathcal{GC}_2 (but not to both). An example of such global constraint is the group constraint. Within the graph invariants this situation is denoted by `vpartition`.

In these situations the graph properties associated with the two graph constraints are also not independent.

In practice the graphs associated with global constraints have a regular structure which comes from the initial graph or from the property of the arc constraints. So, in addition to graph invariants that hold for any graph, we want also tighter graph invariants that hold for specific graph classes. The next section introduces the graph classes we consider, while the two other sections give the graph invariants on one and two graphs.

3.2.1 Graph classes

By definition, a graph invariant has to hold for any digraph. For instance, we have the graph invariant $\mathbf{NARC} \leq \mathbf{NVERTEX}^2$, which relates the number of arcs and the number of vertices of any digraph. This invariant is sharp since the equality is reached for a clique. However, by considering the structure of a digraph, we can get sharper invariants. For instance, if our digraph is a subset of an elementary path (e.g. we use the *PATH* arc generator depicted by Figure 1.4) we have that $\mathbf{NARC} \leq \mathbf{NVERTEX} - 1$, which is a tighter bound of the maximum number of arcs since $\mathbf{NVERTEX} - 1 < \mathbf{NVERTEX}^2$. For this reason, we consider recurring graph classes that show up for different global constraints of the catalog. For a given global constraint, a graph class specifies a general property that holds on its final digraph. We list the different graph classes and, for each of them, we point to some global constraints that fit in that class. Finding all the global constraints corresponding to a given graph class can be done by looking into the list of keywords (see Section 2.5 page 62).

- `acyclic`: graph constraint for which the final graph doesn't have any circuit.
- `apartition`: constraint defined by two graph constraints having the same initial graph, where each arc of the initial graph belongs to one of the final graph (but not to both).
- `bipartite`: graph constraint for which the final graph is bipartite.
- `consecutive_loops_are_connected`: denotes the fact that the graph constraints of a global constraint use only the *PATH* and the *LOOP* arc generators and that their final graphs do not contain consecutive vertices that have a loop and that are not connected together by an arc.
- `equivalence`: graph constraint for which the final graph is reflexive, symmetric and transitive.
- `no_loop`: graph constraint for which the final graph doesn't have any loop.
- `one_succ`: graph constraint for which all the vertices of the initial graph belong to the final graph and for which all vertices of the final graph have exactly one successor.
- `symmetric`: graph constraint for which the final graph is symmetric.
- `vpartition`: constraint defined by two graph constraints having the same initial graph, where each vertex of the initial graph belongs to one of the final graph (but not to both).

In addition, we also consider graph constraints such that their final graphs is a subset of the graph generated by the arc generators:

- *CHAIN*,
- *CIRCUIT*,
- *CLIQUE*,
- *CLIQUE*(Comparison),
- *GRID*,
- *LOOP*,
- *PATH*,
- *PRODUCT*,
- *PRODUCT*(Comparison),
- *SYMMETRIC_PRODUCT*,
- *SYMMETRIC_PRODUCT*(Comparison),

where Comparison is one of the following comparison operators $\leq, \geq, <, >, =, \neq$.

3.2.2 Format of an invariant

As we previously saw, we have graph invariants that hold for any digraph as well as tighter graph invariants for specific graph classes. As a consequence, we partition the database in groups of graph invariants. A *group of graph invariants* corresponds to several invariants such that all invariants relate the same subset of graph characteristics and such that all invariants are variations of the first invariant of the group taking into accounts the graph class. Therefore, the first invariant of a group has no precondition, while all other invariants have a non-empty precondition that characterizes the graph class for which they hold.

EXAMPLE: As a first example consider the group of invariants denoted by Proposition 64, which relate the number of arcs *NARC* with the number of vertices of the smallest and largest connected component (i.e. *MIN_NCC* and *MAX_NCC*).

$$\text{MIN_NCC} \neq \text{MAX_NCC} \Rightarrow \text{NARC} \geq \text{MIN_NCC} + \text{MAX_NCC} - 2 + (\text{MIN_NCC} = 1)$$

$$\text{equivalence : MIN_NCC} \neq \text{MAX_NCC} \Rightarrow$$

$$\text{NARC} \geq \text{MIN_NCC}^2 + \text{MAX_NCC}^2$$

On the one hand, since the first rule has no precondition it corresponds to a general graph invariant. On the other hand the second rule specifies a tighter condition (since $\text{MIN_NCC}^2 + \text{MAX_NCC}^2$ is greater than or equal to $\text{MIN_NCC} + \text{MAX_NCC} - 2 + (\text{MIN_NCC} = 1)$), which only holds for a final graph, which is reflexive, symmetric and transitive.

EXAMPLE: As a second example, consider the following group of invariants corresponding to Proposition 49, which relate the number of arcs \mathbf{NARC} to the number of vertices $\mathbf{NVERTEX}$ according to the arc generator (see Figure 1.4) used for generating the initial digraph:

$$\begin{aligned} & \mathbf{NARC} \leq \mathbf{NVERTEX}^2 \\ \text{arc_gen} = \text{CIRCUIT} : & \mathbf{NARC} \leq \mathbf{NVERTEX} \\ \text{arc_gen} = \text{CHAIN} : & \mathbf{NARC} \leq 2 \cdot \mathbf{NVERTEX} - 2 \\ \text{arc_gen} = \text{CLIQUE}(\leq) : & \mathbf{NARC} \leq \frac{\mathbf{NVERTEX} \cdot (\mathbf{NVERTEX} + 1)}{2} \\ \text{arc_gen} = \text{CLIQUE}(\geq) : & \mathbf{NARC} \leq \frac{\mathbf{NVERTEX} \cdot (\mathbf{NVERTEX} + 1)}{2} \\ \text{arc_gen} = \text{CLIQUE}(<) : & \mathbf{NARC} \leq \frac{\mathbf{NVERTEX} \cdot (\mathbf{NVERTEX} - 1)}{2} \\ \text{arc_gen} = \text{CLIQUE}(>) : & \mathbf{NARC} \leq \frac{\mathbf{NVERTEX} \cdot (\mathbf{NVERTEX} - 1)}{2} \\ \text{arc_gen} = \text{CLIQUE}(\neq) : & \mathbf{NARC} \leq \mathbf{NVERTEX}^2 - \mathbf{NVERTEX} \\ \text{arc_gen} = \text{CYCLE} : & \mathbf{NARC} \leq 2 \cdot \mathbf{NVERTEX} \\ \text{arc_gen} = \text{PATH} : & \mathbf{NARC} \leq \mathbf{NVERTEX} - 1 \end{aligned}$$

3.2.3 Using the database of invariants

The purpose of this section is to provide a set of graph invariants, each invariant relating a given set of graph characteristics. Once we have these graph invariants we can use them systematically by applying the following steps:

- For a given graph constraint we extract all the graph characteristics occurring in its description. This can be done automatically by scanning the corresponding graph properties. Let \mathcal{GC} denote this subset of graph characteristics. For each graph characteristic gc of \mathcal{GC} we check if we have a graph property of the form $gc = var$ where var is a domain variable. If this is the case we record the pair (gc, var) ; if not, we create a new domain variable var and also record the pair (gc, var) .
- We then search for all groups of graph invariants involving a subset of the previous graph characteristics \mathcal{GC} . For each selected group we filter out those graph invariants for which the preconditions are not compatible with the graph class of the graph constraint under consideration. In each group we finally keep those invariants that have the maximum number of preconditions (i.e. the most specialized graph invariants).
- Finally we state all the previous collected graph invariants as implied constraints. This is achieved by using the variables associated with each graph characteristic.

EXAMPLE: We continue with the example of the group constraint and its first graph constraint. The steps for creating the implied constraints are:

- We first extract the graph characteristics **NCC**, **MIN_NCC**, **MAX_NCC** and **NVERTEX** from the first graph constraint of the group constraint. Since all the graph properties attached to the previous graph characteristics have the form $gc = var$ we extract the corresponding domain variables and get the following pairs (**NCC**, **NGROUP**), (**MIN_NCC**, **MIN_SIZE**), (**MAX_NCC**, **MAX_SIZE**) and (**NVERTEX**, **NVAL**).
- We search for all groups of graph invariants involving the graph characteristics **NCC**, **MIN_NCC**, **MAX_NCC** and **NVERTEX** and filter out the irrelevant graph invariants that can't be applied on the graph class associated with the group constraint.
- We state all the previous invariants by substituting each graph characteristics by its corresponding variable, which leads to a set of implied constraints.

3.2.4 The database of graph invariants

For each combination of graph characteristics we give the number of graph invariants we currently have. The items are sorted first in increasing number of graph characteristics of the invariant, second in alphabetic order on the name of the characteristics. All graph invariants assume a digraph for which each vertex has at least one arc. For some propositions, a figure depicts the corresponding final graph, which minimizes or maximizes a given graph characteristics. The propositions of this section and their corresponding proofs use the notations introduced in Section 1.2.2 page 31.

- Graph invariants involving one graph characteristics of a final graph:
 - **MAX_NCC**: 1 (see Proposition 1),
 - **MAX_NSCC**: 2 (see Propositions 2 and 3),
 - **MIN_NCC**: 1 (see Proposition 4),
 - **MIN_NSCC**: 2 (see Propositions 5 and 6),
 - **NARC**: 1 (see Proposition 7),
 - **NCC**: 2 (see Propositions 8 and 9),
 - **NSCC**: 1 (see Proposition 10),
 - **NSINK**: 1 (see Proposition 11),
 - **NSOURCE**: 1 (see Proposition 12),
 - **NVERTEX**: 1 (see Proposition 13).
- Graph invariants involving two graph characteristics of a final graph:
 - **MAX_NCC**, **MAX_NSCC**: 2 (see Propositions 14 and 15),
 - **MAX_NCC**, **MIN_NCC**: 2 (see Propositions 16 and 17),
 - **MAX_NCC**, **NARC**: 2 (see Propositions 18 and 19),
 - **MAX_NCC**, **NSINK**: 2 (see Propositions 20 and 21),
 - **MAX_NCC**, **NSOURCE**: 2 (see Propositions 22 and 23),

- MAX_NCC, NVERTEX: 2 (see Propositions 24 and 25),
 - MAX_NSCC, MIN_NSCC: 2 (see Propositions 26 and 27),
 - MAX_NSCC, NARC: 2 (see Propositions 28 and 29),
 - MAX_NSCC, NVERTEX: 2 (see Propositions 30 and 31),
 - MIN_NCC, MIN_NSCC: 2 (see Propositions 32 and 33),
 - MIN_NCC, NARC: 2 (see Propositions 34 and 35),
 - MIN_NCC, NCC: 1 (see Proposition 36),
 - MIN_NCC, NVERTEX: 3 (see Propositions 37, 38 and 39),
 - MIN_NSCC, NARC: 2 (see Propositions 40 and 41),
 - MIN_NSCC, NVERTEX: 2 (see Propositions 42 and 43),
 - NARC, NCC: 2 (see Propositions 44 and 45),
 - NARC, NSCC: 2 (see Propositions 46 and 47),
 - NARC, NVERTEX: 4 (see Propositions 48, 49, 50 and 51),
 - NCC, NSCC: 2 (see Propositions 52 and 53),
 - NCC, NVERTEX: 3 (see Propositions 54 and 55 and 56),
 - NSCC, NVERTEX: 3 (see Propositions 57, 58 and 59),
 - NSINK, NVERTEX: 2 (see Propositions 60 and 61),
 - NSOURCE, NVERTEX: 2 (see Propositions 62 and 63).
- Graph invariants involving three graph characteristics of a final graph:
 - MAX_NCC, MIN_NCC, NARC: 1 (see Proposition 64),
 - MAX_NCC, MIN_NCC, NCC: 1 (see Proposition 65),
 - MAX_NCC, MIN_NCC, NVERTEX: 5 (see Propositions 66, 67, 68, 69 and 70),
 - MAX_NCC, NARC, NCC: 2 (see Propositions 71 and 72),
 - MAX_NCC, NARC, NVERTEX: 2 (see Propositions 73 and 74),
 - MAX_NCC, NCC, NVERTEX: 2 (see Propositions 75 and 76),
 - MAX_NSCC, MIN_NSCC, NARC: 1 (see Proposition 77),
 - MAX_NSCC, MIN_NSCC, NSCC: 1 (see Proposition 78),
 - MAX_NSCC, MIN_NSCC, NVERTEX: 2 (see Propositions 79 and 80),
 - MAX_NSCC, NSCC, NVERTEX: 2 (see Propositions 81 and 82),
 - MIN_NCC, NARC, NVERTEX: 2 (see Propositions 83 and 84),
 - MIN_NCC, NCC, NVERTEX: 1 (see Proposition 85),
 - MIN_NSCC, NARC, NVERTEX: 1 (see Proposition 86),
 - MIN_NSCC, NSCC, NVERTEX: 1 (see Proposition 87),
 - NARC, NCC, NVERTEX: 2 (see Propositions 88 and 89),
 - NARC, NSCC, NVERTEX: 3 (see Propositions 90, 91 and 92),
 - NARC, NSINK, NVERTEX: 2 (see Propositions 93 and 94),
 - NARC, NSOURCE, NVERTEX: 2 (see Propositions 95 and 96),
 - NSINK, NSOURCE, NVERTEX: 1 (see Proposition 97).
 - Graph invariants involving four graph characteristics of a final graph:
 - MAX_NCC, MIN_NCC, NARC, NCC: 2 (see Propositions 98 and 99),

- MAX_NCC , MIN_NCC , NCC , NVERTEX : 2 (see Propositions 100 and 101),
 - MAX_NSCC , MIN_NSCC , NARC , NSCC : 2 (see Propositions 102 and 103),
 - MAX_NSCC , MIN_NSCC , NSCC , NVERTEX : 2 (see Propositions 104 and 105),
 - MIN_NCC , NARC , NCC , NVERTEX : 1 (see Proposition 106),
 - NARC , NCC , NSCC , NVERTEX : 2 (see Propositions 107 and 108),
 - NARC , NSINK , NSOURCE , NVERTEX : 1 (see Proposition 109).
- Graph invariants involving five graph characteristics of a final graph:
 - MAX_NCC , MIN_NCC , NARC , NCC , NVERTEX : 1 (see Proposition 110),
 - MIN_NCC , NARC , NCC , NSCC , NVERTEX : 1 (see Proposition 111).
- Graph invariants relating two characteristics of two final graphs:
 - MAX_NCC_1 , NCC_2 : 1 (see Proposition 112),
 - MAX_NCC_2 , NCC_1 : 1 (see Proposition 113),
 - MIN_NCC_1 , NCC_2 : 1 (see Proposition 114),
 - MIN_NCC_2 , NCC_1 : 1 (see Proposition 115),
 - NARC_1 , NARC_2 : 1 (see Proposition 116),
 - NCC_1 , NCC_2 : 2 (see Propositions 117 and 118),
 - NVERTEX_1 , NVERTEX_2 : 1 (see Proposition 119).
- Graph invariants relating three characteristics of two final graphs:
 - MAX_NCC_1 , MIN_NCC_1 , MIN_NCC_2 : 2 (see Propositions 120 and 121),
 - MAX_NCC_2 , MIN_NCC_2 , MIN_NCC_1 : 2 (see Propositions 122 and 123),
 - MIN_NCC_1 , NARC_2 , NCC_1 : 1 (see Proposition 124),
 - MIN_NCC_2 , NARC_1 , NCC_2 : 1 (see Proposition 125).
- Graph invariants relating four characteristics of two final graphs:
 - MAX_NCC_1 , MIN_NCC_1 , MIN_NCC_2 , NCC_1 : 2 (see Propositions 126 and 127),
 - MAX_NCC_2 , MIN_NCC_2 , MIN_NCC_1 , NCC_2 : 2 (see Propositions 128 and 129).
- Graph invariants relating five characteristics of two final graphs:
 - MAX_NCC_1 , MAX_NCC_2 , MIN_NCC_1 , MIN_NCC_2 , NCC_1 : 7 (see Propositions 130, 131, 132, 133, 134, 135 and 136).
 - MAX_NCC_1 , MAX_NCC_2 , MIN_NCC_1 , MIN_NCC_2 , NCC_2 : 7 (see Propositions 137, 138, 139, 140, 141, 142 and 143).
- Graph invariants relating six characteristics of two final graphs:
 - MAX_NCC_1 , MAX_NCC_2 , MIN_NCC_1 , MIN_NCC_2 , NCC_1 , NCC_2 : 2 (see Propositions 144 and 145).

Graph invariants involving one characteristic of a final graph**MAX_NCC****Proposition 1.**

$$\text{no_loop} : \text{MAX_NCC} \neq 1 \quad (3.1)$$

Proof. Since we don't have any loop, a non-empty connected component has at least two vertices. \square

MAX_NSCC**Proposition 2.**

$$\text{acyclic} : \text{MAX_NSCC} \leq 1 \quad (3.2)$$

Proof. Since we don't have any circuit, a non-empty strongly connected component consists of one single vertex. \square

Proposition 3.

$$\text{no_loop} : \text{MAX_NSCC} \neq 1 \quad (3.3)$$

Proof. Since we don't have any loop, a non-empty strongly connected component has at least two vertices. \square

MIN_NCC**Proposition 4.**

$$\text{no_loop} : \text{MIN_NCC} \neq 1 \quad (3.4)$$

Proof. Since we don't have any loop, a non-empty connected component has at least two vertices. \square

MIN_NSCC**Proposition 5.**

$$\text{acyclic} : \text{MIN_NSCC} \leq 1 \quad (3.5)$$

Proof. Since we don't have any circuit, a non-empty strongly connected component consists of one single vertex. \square

Proposition 6.

$$\text{no_loop} : \text{MIN_NSCC} \neq 1 \quad (3.6)$$

Proof. Since we don't have any loop, a non-empty strongly connected component has at least two vertices. \square

NARC**Proposition 7.**

$$\text{one_succ} : \text{NARC} = \text{NVERTEX}_{\text{INITIAL}} \quad (3.7)$$

Proof. By definition of `one_succ`. \square

NCC**Proposition 8.**

$$\text{no_loop} : 2 \cdot \text{NCC} \leq \text{NVERTEX}_{\text{INITIAL}} \quad (3.8)$$

Proof. By definition of `no_loop`, each connected component has at least two vertices. \square

Proposition 9.

$$\text{consecutive_loops_are_connected} : 2 \cdot \text{NCC} \leq \text{NVERTEX}_{\text{INITIAL}} + 1 \quad (3.9)$$

Proof. By definition of `consecutive_loops_are_connected`. \square

NSCC**Proposition 10.**

$$\text{no_loop} : 2 \cdot \text{NSCC} \leq \text{NVERTEX}_{\text{INITIAL}} \quad (3.10)$$

Proof. By definition of `no_loop`, each strongly connected component has at least two vertices. \square

NSINK**Proposition 11.**

$$\text{symmetric} : \text{NSINK} = 0 \quad (3.11)$$

Proof. Since we don't have any isolated vertex. \square

NSOURCE**Proposition 12.**

$$\text{symmetric} : \text{NSOURCE} = 0 \quad (3.12)$$

Proof. Since we don't have any isolated vertex. \square

NVERTEX**Proposition 13.**

$$\text{one_succ} : \text{NVERTEX} = \text{NVERTEX}_{\text{INITIAL}} \quad (3.13)$$

Proof. By definition of `one_succ`. \square

Graph invariants involving two characteristics of a final graph**MAX_NCC, MAX_NSCC****Proposition 14.**

$$\text{MAX_NCC} = 0 \Leftrightarrow \text{MAX_NSCC} = 0 \quad (3.14)$$

Proof. By definition of **MAX_NCC** and of **MAX_NSCC**. □

Proposition 15.

$$\text{MAX_NSCC} \leq \text{MAX_NCC} \quad (3.15)$$

Proof. **MAX_NSCC** is a lower bound of the size of the largest connected component since the largest strongly connected component is for sure included within a connected component. □

MAX_NCC, MIN_NCC**Proposition 16.**

$$\text{MAX_NCC} = 0 \Leftrightarrow \text{MIN_NCC} = 0 \quad (3.16)$$

Proof. By definition of **MAX_NCC** and of **MIN_NCC**. □

Proposition 17.

$$\text{MIN_NCC} \leq \text{MAX_NCC} \quad (3.17)$$

Proof. By definition of **MIN_NCC** and of **MAX_NCC**. □

MAX_NCC, NARC**Proposition 18.**

$$\text{MAX_NCC} = 0 \Leftrightarrow \text{NARC} = 0 \quad (3.18)$$

Proof. By definition of **MAX_NCC** and of **NARC**. □

Proposition 19.

$$\text{MAX_NCC} > 0 \Rightarrow \text{NARC} \geq \max(1, \text{MAX_NCC} - 1) \quad (3.19)$$

$$\text{symmetric} : \text{MAX_NCC} > 0 \Rightarrow \text{NARC} \geq \max(1, 2 \cdot \text{MAX_NCC} - 2) \quad (3.20)$$

$$\text{equivalence} : \text{NARC} \geq \text{MAX_NCC}^2 \quad (3.21)$$

$$\text{arc_gen} = \text{PATH} : \text{NARC} \geq \text{MAX_NCC} - 1 \quad (3.22)$$

Proof.

(3.19) **MAX_NCC** - 1 arcs are needed to connect **MAX_NCC** vertices that belong to a given connected component containing at least two vertices. And one arc is required for a connected component containing one single vertex.

(3.20) Similarly, when the graph is symmetric, $2 \cdot \text{MAX_NCC} - 2$ arcs are needed to connect **MAX_NCC** vertices that belong to a given connected component containing at least two vertices.

(3.21) Finally, when the graph is reflexive, symmetric and transitive, MAX_NCC^2 arcs are needed to connect MAX_NCC vertices that belong to a given connected component.

(3.22) When the initial graph corresponds to a path, the minimum number of arcs of a connected component involving n vertices is equal to $n - 1$. □

MAX_NCC, NSINK

Proposition 20.

$$\text{MAX_NCC} = 0 \Rightarrow \text{NSINK} = 0 \quad (3.23)$$

Proof. By definition of MAX_NCC and of NSINK . □

Proposition 21.

$$\text{NSINK} \geq 1 \Rightarrow \text{MAX_NCC} \geq 2 \quad (3.24)$$

Proof. Since we don't have any isolated vertex a sink is connected to at least one other vertex. Therefore, if the graph has a sink, there exists at least one connected component with at least two vertices. □

MAX_NCC, NSOURCE

Proposition 22.

$$\text{MAX_NCC} = 0 \Rightarrow \text{NSOURCE} = 0 \quad (3.25)$$

Proof. By definition of MAX_NCC and of NSOURCE . □

Proposition 23.

$$\text{NSOURCE} \geq 1 \Rightarrow \text{MAX_NCC} \geq 2 \quad (3.26)$$

Proof. Since we don't have any isolated vertex a source is connected to at least one other vertex. Therefore, if the graph has a source, there exists at least one connected component with at least two vertices. □

MAX_NCC, NVERTEX

Proposition 24.

$$\text{MAX_NCC} = 0 \Leftrightarrow \text{NVERTEX} = 0 \quad (3.27)$$

Proof. By definition of MAX_NCC and of NVERTEX . □

Proposition 25.

$$\text{NVERTEX} \geq \text{MAX_NCC} \quad (3.28)$$

Proof. By definition of MAX_NCC . □

MAX_NSCC, MIN_NSCC**Proposition 26.**

$$\text{MAX_NSCC} = 0 \Leftrightarrow \text{MIN_NSCC} = 0 \quad (3.29)$$

Proof. By definition of MAX_NSCC and of MIN_NSCC. \square

Proposition 27.

$$\text{MIN_NSCC} \leq \text{MAX_NSCC} \quad (3.30)$$

Proof. By definition of MIN_NSCC and of MAX_NSCC. \square

MAX_NSCC, NARC**Proposition 28.**

$$\text{MAX_NSCC} = 0 \Leftrightarrow \text{NARC} = 0 \quad (3.31)$$

Proof. By definition of MAX_NSCC and of NARC. \square

Proposition 29.

$$\text{NARC} \geq \text{MAX_NSCC} \quad (3.32)$$

$$\text{symmetric} : \text{NARC} \geq 2 \cdot \text{MAX_NSCC} \quad (3.33)$$

$$\text{equivalence} : \text{NARC} \geq \text{MAX_NSCC}^2 \quad (3.34)$$

Proof. (3.32) In a strongly connected component at least one arc has to leave each vertex. Since we have at least one strongly connected component of MAX_NSCC vertices this leads to the previous inequality. \square

MAX_NSCC, NVERTEX**Proposition 30.**

$$\text{MAX_NSCC} = 0 \Leftrightarrow \text{NVERTEX} = 0 \quad (3.35)$$

Proof. By definition of MAX_NSCC and of NVERTEX. \square

Proposition 31.

$$\text{NVERTEX} \geq \text{MAX_NSCC} \quad (3.36)$$

Proof. By definition of MAX_NSCC. \square

MIN_NCC, MIN_NSCC**Proposition 32.**

$$\text{MIN_NCC} = 0 \Leftrightarrow \text{MIN_NSCC} = 0 \quad (3.37)$$

Proof. By definition of MIN_NCC and of MIN_NSCC. \square

Proposition 33.

$$\text{MIN_NCC} \geq \text{MIN_NSCC} \quad (3.38)$$

Proof. By construction MIN_NCC is an upper bound of the number of vertices of the smallest strongly connected component. \square

MIN_NCC, NARC

Proposition 34.

$$\text{MIN_NCC} = 0 \Leftrightarrow \text{NARC} = 0 \quad (3.39)$$

Proof. By definition of **MIN_NCC** and of **NARC**. \square

Proposition 35.

$$\text{MIN_NCC} > 0 \Rightarrow \text{NARC} \geq \max(1, \text{MIN_NCC} - 1) \quad (3.40)$$

$$\text{symmetric} : \text{MIN_NCC} > 0 \Rightarrow \text{NARC} \geq \max(1, 2 \cdot \text{MIN_NCC} - 2) \quad (3.41)$$

$$\text{equivalence} : \text{NARC} \geq \text{MIN_NCC}^2 \quad (3.42)$$

$$\text{arc_gen} = \text{PATH} : \text{NARC} \geq \text{MIN_NCC} - 1 \quad (3.43)$$

Proof. Similar to Proposition 19. \square

MIN_NCC, NCC

Proposition 36.

$$\text{consecutive_loops_are_connected} : (\text{MIN_NCC} + 1) \cdot \text{NCC} \leq \text{NVERTEX}_{\text{INITIAL}} + 1 \quad (3.44)$$

Proof. By definition of **consecutive_loops_are_connected**. \square

MIN_NCC, NVERTEX

Proposition 37.

$$\text{MIN_NCC} = 0 \Leftrightarrow \text{NVERTEX} = 0 \quad (3.45)$$

Proof. By definition of **MIN_NCC** and of **NVERTEX**. \square

Proposition 38.

$$\text{NVERTEX} \geq \text{MIN_NCC} \quad (3.46)$$

Proof. By definition of **MIN_NCC**. \square

Proposition 39.

$$\text{MIN_NCC} \notin \left[\min \left(\left\lfloor \frac{\text{NVERTEX}}{2} \right\rfloor, \left\lfloor \frac{\text{NVERTEX}_{\text{INITIAL}} - 1}{2} \right\rfloor \right) + 1, \text{NVERTEX} - 1 \right] \quad (3.47)$$

Proof. On the one hand, if $\text{NCC} \leq 1$, we have that $\text{MIN_NCC} \geq \text{NVERTEX}$. On the other hand, if $\text{NCC} > 1$, we have that $\text{MIN_NCC} + \text{MIN_NCC} \leq \text{NVERTEX}$ and that $\text{MIN_NCC} + \text{MIN_NCC} + 1 \leq \text{NVERTEX}_{\text{INITIAL}}$, which by isolating **MIN_NCC** and by grouping the two inequalities leads to $\text{MIN_NCC} \leq \min \left(\left\lfloor \frac{\text{NVERTEX}}{2} \right\rfloor, \left\lfloor \frac{\text{NVERTEX}_{\text{INITIAL}} - 1}{2} \right\rfloor \right)$. The result follows. \square

MIN_NSCC, NARC**Proposition 40.**

$$\text{MIN_NSCC} = 0 \Leftrightarrow \text{NARC} = 0 \quad (3.48)$$

Proof. By definition of **MIN_NSCC** and of **NARC**. □**Proposition 41.**

$$\text{NARC} \geq \text{MIN_NSCC} \quad (3.49)$$

$$\text{symmetric} : \text{NARC} \geq 2 \cdot \text{MIN_NSCC} \quad (3.50)$$

$$\text{equivalence} : \text{NARC} \geq \text{MIN_NSCC}^2 \quad (3.51)$$

Proof. Similar to Proposition 29. □**MIN_NSCC, NVERTEX****Proposition 42.**

$$\text{MIN_NSCC} = 0 \Leftrightarrow \text{NVERTEX} = 0 \quad (3.52)$$

Proof. By definition of **MIN_NSCC** and of **NVERTEX**. □**Proposition 43.**

$$\text{NVERTEX} \geq \text{MIN_NSCC} \quad (3.53)$$

Proof. By definition of **MIN_NSCC**. □**NARC, NCC****Proposition 44.**

$$\text{NARC} = 0 \Leftrightarrow \text{NCC} = 0 \quad (3.54)$$

Proof. By definition of **NARC** and of **NCC**. □**Proposition 45.**

$$\text{NARC} \geq \text{NCC} \quad (3.55)$$

Proof. Each connected component contains at least one arc (since, by hypothesis, each vertex has at least one arc). □**NARC, NSCC****Proposition 46.**

$$\text{NARC} = 0 \Leftrightarrow \text{NSCC} = 0 \quad (3.56)$$

Proof. By definition of **NARC** and of **NSCC**. □**Proposition 47.**

$$\text{NARC} \geq \text{NSCC} \quad (3.57)$$

$$\text{no_loop} : \text{NARC} \geq 2 \cdot \text{NSCC} \quad (3.58)$$

Proof. 3.57 (respectively 3.58) holds since each strongly connected component contains at least one (respectively two) arc(s). □

NARC, NVERTEX

Proposition 48.

$$\text{NARC} = 0 \Leftrightarrow \text{NVERTEX} = 0 \quad (3.59)$$

Proof. By definition of NARC and of NVERTEX. \square **Proposition 49.**

$$\text{NARC} \leq \text{NVERTEX}^2 \quad (3.60)$$

$$\text{arc_gen} = \text{CIRCUIT} : \text{NARC} \leq \text{NVERTEX} \quad (3.61)$$

$$\text{arc_gen} = \text{CHAIN} : \text{NARC} \leq 2 \cdot \text{NVERTEX} - 2 \quad (3.62)$$

$$\text{arc_gen} = \text{CLIQUE}(\leq) : \text{NARC} \leq \frac{\text{NVERTEX} \cdot (\text{NVERTEX} + 1)}{2} \quad (3.63)$$

$$\text{arc_gen} = \text{CLIQUE}(\geq) : \text{NARC} \leq \frac{\text{NVERTEX} \cdot (\text{NVERTEX} + 1)}{2} \quad (3.64)$$

$$\text{arc_gen} = \text{CLIQUE}(<) : \text{NARC} \leq \frac{\text{NVERTEX} \cdot (\text{NVERTEX} - 1)}{2} \quad (3.65)$$

$$\text{arc_gen} = \text{CLIQUE}(>) : \text{NARC} \leq \frac{\text{NVERTEX} \cdot (\text{NVERTEX} - 1)}{2} \quad (3.66)$$

$$\text{arc_gen} = \text{CLIQUE}(\neq) : \text{NARC} \leq \text{NVERTEX}^2 - \text{NVERTEX} \quad (3.67)$$

$$\text{arc_gen} = \text{CYCLE} : \text{NARC} \leq 2 \cdot \text{NVERTEX} \quad (3.68)$$

$$\text{arc_gen} = \text{PATH} : \text{NARC} \leq \text{NVERTEX} - 1 \quad (3.69)$$

Proof. 3.60 holds since each vertex of a digraph can have at most NVERTEX successors. The next items correspond to the maximum number of arcs that can be achieved according to a specific arc generator. \square **Proposition 50.**

$$2 \cdot \text{NARC} \geq \text{NVERTEX} \quad (3.70)$$

Proof. By induction on the number of vertices of a graph G :

1. If NVERTEX(G) is equal to 1 or 2 Proposition 50 holds.
2. Assume that NVERTEX(G) ≥ 3 .
 - Assume there exists a vertex v such that, if we remove v , we don't create any isolated vertex in the remaining graph. We have $\text{NARC}(G) \geq \text{NARC}(G - v) + 1$. Thus $2 \cdot \text{NARC}(G) \geq 2 \cdot \text{NARC}(G - v) + 1$. Since by induction hypothesis $2 \cdot \text{NARC}(G - v) \geq \text{NVERTEX}(G - v) = \text{NVERTEX}(G) - 1$ the result holds.

- Otherwise, all the connected components of G are reduced to two elements with only one arc. We remove one of such connected component (v, w) .

Thus $\text{NARC}(G) = \text{NARC}(G - \{v, w\}) + 1$. As by induction hypothesis, $2 \cdot \text{NARC}(G - \{v, w\}) \geq \text{NVERTEX}(G - \{v, w\}) = \text{NVERTEX}(G) - 2$ the result holds. □

Proposition 51.

$$\text{arc_gen} = \text{LOOP} : \text{NARC} = \text{NVERTEX} \quad (3.71)$$

Proof. From the definition of LOOP . □

NCC, NSCC

Proposition 52.

$$\text{NCC} = 0 \Leftrightarrow \text{NSCC} = 0 \quad (3.72)$$

Proof. By definition of NCC and of NSCC . □

Proposition 53.

$$\text{NCC} \leq \text{NSCC} \quad (3.73)$$

Proof. Holds since each connected component contains at least one strongly connected component. □

NCC, NVERTEX

Proposition 54.

$$\text{NCC} = 0 \Leftrightarrow \text{NVERTEX} = 0 \quad (3.74)$$

Proof. By definition of NCC and of NVERTEX . □

Proposition 55.

$$\text{NCC} \leq \text{NVERTEX} \quad (3.75)$$

$$\text{no_loop} : 2 \cdot \text{NCC} \leq \text{NVERTEX} \quad (3.76)$$

Proof. 3.75 (respectively 3.76) holds since each connected component contains at least one (respectively two) vertex. □

Proposition 56.

$$\begin{aligned} & \text{vpartition} \wedge \text{consecutive_loops_are_connected} : \\ & \text{NVERTEX} \leq \text{NVERTEX}_{\text{INITIAL}} - (\text{NCC} - 1) \end{aligned} \quad (3.77)$$

Proof. Holds since between two "consecutive" connected components of the initial graph there is at least one vertex, which is missing. □

NSCC, NVERTEX**Proposition 57.**

$$\text{NSCC} = 0 \Leftrightarrow \text{NVERTEX} = 0 \quad (3.78)$$

Proof. By definition of NSCC and of NVERTEX. \square

Proposition 58.

$$\text{NSCC} \leq \text{NVERTEX} \quad (3.79)$$

$$\text{no_loop} : 2 \cdot \text{NSCC} \leq \text{NVERTEX} \quad (3.80)$$

Proof. 3.79 (respectively 3.80) holds since each strongly connected component contains at least one (respectively 2) vertex. \square

Proposition 59.

$$\text{acyclic} : \text{NSCC} = \text{NVERTEX} \quad (3.81)$$

Proof. In a directed acyclic graph we have that each vertex corresponds to a strongly connected component involving only that vertex. \square

NSINK, NVERTEX**Proposition 60.**

$$\text{NVERTEX} = 0 \Rightarrow \text{NSINK} = 0 \quad (3.82)$$

Proof. By definition of NVERTEX and of NSINK. \square

Proposition 61.

$$\text{NVERTEX} > 0 \Rightarrow \text{NSINK} < \text{NVERTEX} \quad (3.83)$$

Proof. Holds since each sink must have a predecessor which cannot be a sink and since each vertex has at least one arc. \square

NSOURCE, NVERTEX**Proposition 62.**

$$\text{NVERTEX} = 0 \Rightarrow \text{NSOURCE} = 0 \quad (3.84)$$

Proof. By definition of NVERTEX and of NSOURCE. \square

Proposition 63.

$$\text{NVERTEX} > 0 \Rightarrow \text{NSOURCE} < \text{NVERTEX} \quad (3.85)$$

Proof. Holds since each source must have a successor which cannot be a source and since each vertex has at least one arc. \square

Graph invariants involving three characteristics of a final graph

MAX_NCC, MIN_NCC, NARC

Proposition 64.

$$\begin{aligned} \text{MIN_NCC} \neq \text{MAX_NCC} \Rightarrow \\ \text{NARC} \geq \text{MIN_NCC} + \text{MAX_NCC} - 2 + (\text{MIN_NCC} = 1) \end{aligned} \quad (3.86)$$

$$\begin{aligned} \text{equivalence : } \text{MIN_NCC} \neq \text{MAX_NCC} \Rightarrow \\ \text{NARC} \geq \text{MIN_NCC}^2 + \text{MAX_NCC}^2 \end{aligned} \quad (3.87)$$

Proof. (3.86) $n - 1$ arcs are needed to connect n ($n > 1$) vertices that all belong to a given connected component. Since we have two connected components which respectively have MIN_NCC and MAX_NCC vertices this leads to the previous inequality. When MIN_NCC is equal to one we need an extra arc. \square

MAX_NCC, MIN_NCC, NCC

Proposition 65.

$$\text{MIN_NCC} \neq \text{MAX_NCC} \Rightarrow \text{NCC} \geq 2 \quad (3.88)$$

Proof. If MIN_NCC and MAX_NCC are different then they correspond for sure to at least two distinct connected components. \square

MAX_NCC, MIN_NCC, NVERTEX

Proposition 66.

$$\text{MIN_NCC} \neq \text{MAX_NCC} \Rightarrow \text{NVERTEX} \geq \text{MIN_NCC} + \text{MAX_NCC} \quad (3.89)$$

Proof. Since we have at least two distinct connected components which respectively have MIN_NCC and MAX_NCC vertices this leads to the previous inequality. \square

Proposition 67.

$$\text{MAX_NCC} \leq \max(\text{MIN_NCC}, \text{NVERTEX} - \max(1, \text{MIN_NCC})) \quad (3.90)$$

Proof. On the one hand, if $\text{NCC} \leq 1$, we have that $\text{MAX_NCC} \leq \text{MIN_NCC}$. On the other hand, if $\text{NCC} > 1$, we have that $\text{NVERTEX} \geq \max(1, \text{MIN_NCC}) + \text{MAX_NCC}$ (i.e. $\text{MAX_NCC} \leq \text{NVERTEX} - \max(1, \text{MIN_NCC})$). The result is obtained by taking the maximum value of the right hand side of the two inequalities. \square

Proposition 68.

$$\text{MIN_NCC} \notin [\text{NVERTEX} - \max(1, \text{MAX_NCC}) + 1, \text{NVERTEX} - 1] \quad (3.91)$$

Proof. On the one hand, if $\text{NCC} \leq 1$, we have that $\text{MIN_NCC} \geq \text{NVERTEX}$. On the other hand, if $\text{NCC} > 1$, we have that $\text{MIN_NCC} + \max(1, \text{MAX_NCC}) \leq \text{NVERTEX}$ (i.e. $\text{MIN_NCC} \leq \text{NVERTEX} - \max(1, \text{MAX_NCC})$). The result follows. \square

Proposition 69.

$$\mathbf{NVERTEX} \notin [\mathbf{MIN_NCC} + 1, \mathbf{MIN_NCC} + \mathbf{MAX_NCC} - 1] \quad (3.92)$$

Proof. On the one hand, if $\mathbf{NCC} \leq 1$, we have that $\mathbf{NVERTEX} \leq \mathbf{MIN_NCC}$. On the other hand, if $\mathbf{NCC} > 1$, we have that $\mathbf{NVERTEX} \geq \mathbf{MIN_NCC} + \mathbf{MAX_NCC}$. Since $\mathbf{MIN_NCC} \leq \mathbf{MIN_NCC} + \mathbf{MAX_NCC}$ the result follows. \square

Proposition 70.

$$\begin{aligned} & \text{if } \overline{\mathbf{MIN_NCC}} > 0 \\ & \text{then } k_{inf} = \left\lfloor \frac{\overline{\mathbf{NVERTEX}} + \overline{\mathbf{MIN_NCC}}}{\overline{\mathbf{MIN_NCC}}} \right\rfloor \text{ else } k_{inf} = 1 \\ \\ & \text{if } \overline{\mathbf{MAX_NCC}} > 0 \\ & \text{then } k_{sup1} = \left\lfloor \frac{\overline{\mathbf{NVERTEX}} - 1}{\overline{\mathbf{MAX_NCC}}} \right\rfloor \text{ else } k_{sup1} = \overline{\mathbf{NVERTEX}} \\ \\ & \text{if } \overline{\mathbf{MAX_NCC}} < \overline{\mathbf{MIN_NCC}} \\ & \text{then } k_{sup2} = \left\lfloor \frac{\overline{\mathbf{MIN_NCC}} - 2}{\overline{\mathbf{MAX_NCC}} - \overline{\mathbf{MIN_NCC}}} \right\rfloor \text{ else } k_{sup2} = \overline{\mathbf{NVERTEX}} \\ \\ & k_{sup} = \min(k_{sup1}, k_{sup2}) \end{aligned}$$

$$\forall k \in [k_{inf}, k_{sup}] : \mathbf{NVERTEX} \notin [k \cdot \mathbf{MAX_NCC} + 1, (k + 1) \cdot \mathbf{MIN_NCC} - 1] \quad (3.93)$$

Proof. We make the proof for $k \in \mathbb{N}$ (the interval $[k_{inf}, k_{sup}]$ is only used for restricting the number of intervals to check). We have that $\mathbf{NVERTEX} \in [k \cdot \mathbf{MIN_NCC}, k \cdot \mathbf{MAX_NCC}]$. A forbidden interval $[k \cdot \mathbf{MAX_NCC} + 1, (k + 1) \cdot \mathbf{MIN_NCC} - 1]$ corresponds to an interval between the end of interval $[k \cdot \mathbf{MIN_NCC}, k \cdot \mathbf{MAX_NCC}]$ and the start of the next interval $[(k + 1) \cdot \mathbf{MIN_NCC}, (k + 1) \cdot \mathbf{MAX_NCC}]$. Since all intervals $[i \cdot \mathbf{MIN_NCC}, i \cdot \mathbf{MAX_NCC}]$ ($i < k$) end before $k \cdot \mathbf{MAX_NCC}$ and since all intervals $[j \cdot \mathbf{MIN_NCC}, j \cdot \mathbf{MAX_NCC}]$ ($j > k$) start after $(k + 1) \cdot \mathbf{MIN_NCC}$, they do not use any value in $[k \cdot \mathbf{MAX_NCC} + 1, (k + 1) \cdot \mathbf{MIN_NCC} - 1]$. \square

$$\boxed{\mathbf{MAX_NCC}, \mathbf{NARC}, \mathbf{NCC}}$$

Proposition 71.

$$\mathbf{NARC} \leq \mathbf{NCC} \cdot \mathbf{MAX_NCC}^2 \quad (3.94)$$

$$\mathbf{arc_gen} = \mathbf{PATH} : \mathbf{NARC} \leq \mathbf{NCC} \cdot (\mathbf{MAX_NCC} - 1) \quad (3.95)$$

Proof. On the one hand, (3.94) holds since the maximum number of arcs is achieved by taking \mathbf{NCC} connected components where each connected component is a clique involving $\mathbf{MAX_NCC}$ vertices. On the other hand, (3.95) holds since a tree of n vertices has $n - 1$ arcs. \square

Proposition 72.

$$\mathbf{NARC} \geq \mathbf{MAX_NCC} + \mathbf{NCC} - 2 \quad (3.96)$$

Proof. The minimum number of arcs is achieved by taking one connected component with $\mathbf{MAX_NCC}$ vertices and $\mathbf{MAX_NCC} - 1$ arcs as well as $\mathbf{NCC} - 1$ connected components with one single vertex and a loop. \square

MAX_NCC, NARC, NVERTEX

Proposition 73.

$$\mathbf{MAX_NCC} > 0 \Rightarrow$$

$$\mathbf{NARC} \leq \mathbf{MAX_NCC}^2 \cdot \left\lfloor \frac{\mathbf{NVERTEX}}{\mathbf{MAX_NCC}} \right\rfloor + (\mathbf{NVERTEX} \bmod \mathbf{MAX_NCC})^2 \quad (3.97)$$

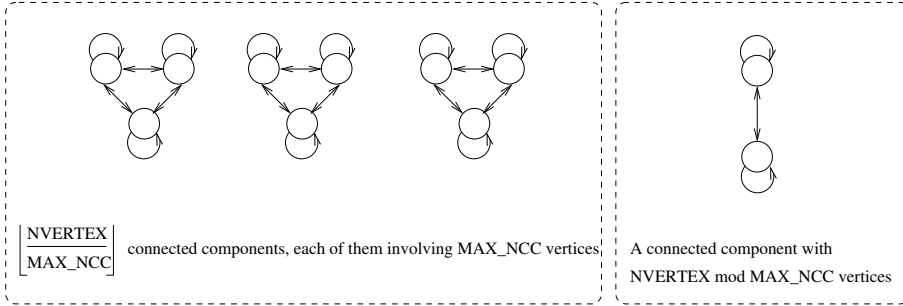


Figure 3.1: Illustration of Proposition 73. A graph that achieves the maximum number of arcs according to the size of the largest connected component as well as to a fixed number of vertices ($\mathbf{MAX_NCC} = 3$, $\mathbf{NVERTEX} = 11$, $\mathbf{NARC} = 3^2 \cdot \lfloor \frac{11}{3} \rfloor + (11 \bmod 3)^2 = 31$)

Proof. We first begin with the following claim:

Let G be a graph such that $V(G) - \mathbf{NCC}(G, \mathbf{MAX_NCC}(G)) * \mathbf{MAX_NCC}(G) \geq \mathbf{MAX_NCC}(G)$, then there exists a graph G' such that $V(G') = V(G)$, $\mathbf{MAX_NCC}(G') = \mathbf{MAX_NCC}(G)$, $\mathbf{NCC}(G', \mathbf{MAX_NCC}(G')) = \mathbf{NCC}(G, \mathbf{MAX_NCC}(G)) + 1$ and $|E(G)| \leq |E(G')|$.

Proof of the claim:

Let $(C_i)_{i \in [n]}$ be the connected components of G on less than $\mathbf{MAX_NCC}(G)$ vertices and such that $|C_i| \geq |C_{i+1}|$. By hypothesis there exists $k \leq n$ such that $|\bigcup_{i=1}^{k-1} C_i| < \mathbf{MAX_NCC}(G)$ and $|\bigcup_{i=1}^k C_i| \geq \mathbf{MAX_NCC}(G)$.

- Either $|\bigcup_{i=1}^k C_i| = \mathbf{MAX_NCC}(G)$, and then with G' such that G' restricted to the $\bigcup_{i=1}^k C_i$ be a complete graph and G' restricted to $V(G) - \bigcup_{i=1}^k C_i$ being exactly G restricted to $V(G) - \bigcup_{i=1}^k C_i$ we obtain the claim.
- Or $|\bigcup_{i=1}^k C_i| > \mathbf{MAX_NCC}(G)$. Then $C_k = C_k^1 \uplus C_k^2$ such that $|\bigcup_{i=1}^{k-1} C_i \cup C_k^1| = \mathbf{MAX_NCC}(G)$ and $|C_k^2| < |C_1|$ (notice that $k \geq 2$). Then with G' such that G' restricted to $(\bigcup_{i=1}^{k-1} C_i) \cup C_k^1$ is a complete graph and G' restricted to $V(G) - ((\bigcup_{i=1}^{k-1} C_i) \cup C_k^1)$ is exactly G restricted to $V(G) - ((\bigcup_{i=1}^{k-1} C_i) \cup C_k^1)$

we obtain the claim.

End of proof of the claim

We prove by induction on $r(G) = \left\lfloor \frac{\text{NVERTEX}(G)}{\text{MAX_NCC}(G)} \right\rfloor - \text{NCC}(G, \text{MAX_NCC}(G))$, where G is any graph. For $r(G) = 0$ the result holds (see Prop 44). Otherwise, since $r(G) > 0$ we have that $V(G) - \text{NCC}(G, \text{MAX_NCC}(G)) * \text{MAX_NCC}(G) \geq \text{MAX_NCC}(G)$, by the previous claim there exists G' with the same number of vertices and the same number of vertices in the largest connected component, such that $r(G') = r(G) - 1$. Consequently the result holds by induction. \square

Proposition 74.

$$\text{NARC} \geq \text{MAX_NCC} - 1 + \left\lfloor \frac{\text{NVERTEX} - \text{MAX_NCC} + 1}{2} \right\rfloor \quad (3.98)$$

Proof. Let G be a graph, let X be a maximal size connected component of G , then we have $G = G[X] \oplus G[V(G) - X]$. On the one hand, as $G[X]$ is connected, by setting $\text{NCC} = 1$ in 3.134 of Proposition 89, we have $|E(G[X])| \geq |X| - 1$, on the other hand, by Proposition 50, $|E(G[V(G) - X])| \geq \left\lfloor \frac{|V(G) - X|}{2} \right\rfloor$. Thus the result follows. \square

MAX_NCC, NCC, NVERTEX

Proposition 75.

$$\text{NVERTEX} \leq \text{NCC} \cdot \text{MAX_NCC} \quad (3.99)$$

Proof. The number of vertices is less than or equal to the number of connected components multiplied by the largest number of vertices in a connected component. \square

Proposition 76.

$$\text{NVERTEX} \geq \text{MAX_NCC} + \max(0, \text{NCC} - 1) \quad (3.100)$$

$$\text{no_loop} : \text{NVERTEX} \geq \text{MAX_NCC} + \max(0, 2 \cdot \text{NCC} - 2) \quad (3.101)$$

Proof. (3.100) The minimum number of vertices according to a fixed number of connected components NCC such that one of the connected component contains MAX_NCC vertices is obtained as follows: We get MAX_NCC vertices from the connected component involving MAX_NCC vertices and one vertex for each remaining connected component. \square

MAX_NSCC, MIN_NSCC, NARC

Proposition 77.

$$\text{MIN_NSCC} \neq \text{MAX_NSCC} \Rightarrow \text{NARC} \geq \text{MIN_NSCC} + \text{MAX_NSCC} \quad (3.102)$$

$$\begin{aligned} \text{equivalence} : \text{MIN_NSCC} \neq \text{MAX_NSCC} \Rightarrow \\ \text{NARC} \geq \text{MIN_NSCC}^2 + \text{MAX_NSCC}^2 \end{aligned} \quad (3.103)$$

Proof. (3.102) In a strongly connected component at least one arc has to leave each arc. Since we have two strongly connected components which respectively have MIN_NSCC and MAX_NSCC vertices this leads to the previous inequality. \square

MAX_NSCC, MIN_NSCC, NSCC

Proposition 78.

$$\text{MIN_NSCC} \neq \text{MAX_NSCC} \Rightarrow \text{NSCC} \geq 2 \quad (3.104)$$

Proof. Follows from the definitions of MIN_NSCC and of MAX_NSCC. \square

MAX_NSCC, MIN_NSCC, NVERTEX

Proposition 79.

$$\text{MIN_NSCC} \neq \text{MAX_NSCC} \Rightarrow \text{NVERTEX} \geq \text{MIN_NSCC} + \text{MAX_NSCC} \quad (3.105)$$

Proof. Since we have at least two distinct strongly connected components which respectively have MIN_NSCC and MAX_NSCC vertices this leads to the previous inequality. \square

Proposition 80.

$$\begin{aligned} & \text{if } \overline{\text{MIN_NSCC}} > 0 \\ & \text{then } k_{inf} = \left\lfloor \frac{\overline{\text{NVERTEX}} + \overline{\text{MIN_NSCC}}}{\overline{\text{MIN_NSCC}}} \right\rfloor \text{ else } k_{inf} = 1 \\ \\ & \text{if } \overline{\text{MAX_NSCC}} > 0 \\ & \text{then } k_{sup_1} = \left\lfloor \frac{\overline{\text{NVERTEX}} - 1}{\overline{\text{MAX_NSCC}}} \right\rfloor \text{ else } k_{sup_1} = \overline{\text{NVERTEX}} \\ \\ & \text{if } \overline{\text{MAX_NSCC}} < \overline{\text{MIN_NSCC}} \\ & \text{then } k_{sup_2} = \left\lfloor \frac{\overline{\text{MIN_NSCC}} - 2}{\overline{\text{MAX_NSCC}} - \overline{\text{MIN_NSCC}}} \right\rfloor \text{ else } k_{sup_2} = \overline{\text{NVERTEX}} \\ \\ & k_{sup} = \min(k_{sup_1}, k_{sup_2}) \end{aligned}$$

$$\forall k \in [k_{inf}, k_{sup}] : \text{NVERTEX} \notin [k \cdot \text{MAX_NSCC} + 1, (k + 1) \cdot \text{MIN_NSCC} - 1] \quad (3.106)$$

Proof. Similar to Proposition 70. \square

MAX_NSCC, NSCC, NVERTEX

Proposition 81.

$$\text{NVERTEX} \leq \text{NSCC} \cdot \text{MAX_NSCC} \quad (3.107)$$

Proof. Since each strongly connected component contains at most MAX_NSCC vertices the total number of vertices is less than or equal to $\text{NSCC} \cdot \text{MAX_NSCC}$. \square

Proposition 82.

$$\text{NVERTEX} \geq \text{MAX_NSCC} + \max(0, \text{NSCC} - 1) \quad (3.108)$$

$$\text{no_loop} : \text{NVERTEX} \geq \text{MAX_NSCC} + \max(0, 2 \cdot \text{NSCC} - 2) \quad (3.109)$$

Proof. (3.108) The minimum number of vertices according to a fixed number of strongly connected components NSCC such that one of them contains MAX_NSCC vertices is equal to $\text{MAX_NSCC} + \max(0, \text{NSCC} - 1)$. \square

MIN_NCC, NARC, NVERTEX

Proposition 83.

$$\text{NARC} \leq \text{MIN_NCC}^2 + (\text{NVERTEX} - \text{MIN_NCC})^2 \quad (3.110)$$

$$\text{arc_gen} = \text{CIRCUIT} : \text{NARC} \leq \text{NVERTEX} - 2 \cdot (\text{MIN_NCC} < \text{NVERTEX}) \quad (3.111)$$

$$\text{arc_gen} = \text{CHAIN} : \text{NARC} \leq \text{NVERTEX} - 2 \cdot (\text{MIN_NCC} < \text{NVERTEX}) \quad (3.112)$$

$$\text{arc_gen} = \text{CLIQUE}(\leq) : \text{NARC} \leq \frac{\text{MIN_NCC} \cdot (\text{MIN_NCC} + 1)}{2} + \frac{(\text{NVERTEX} - \text{MIN_NCC}) \cdot (\text{NVERTEX} - \text{MIN_NCC} + 1)}{2} \quad (3.113)$$

$$\text{arc_gen} = \text{CLIQUE}(\geq) : \text{NARC} \leq \frac{\text{MIN_NCC} \cdot (\text{MIN_NCC} + 1)}{2} + \frac{(\text{NVERTEX} - \text{MIN_NCC}) \cdot (\text{NVERTEX} - \text{MIN_NCC} + 1)}{2} \quad (3.114)$$

$$\text{arc_gen} = \text{CLIQUE}(<) : \text{NARC} \leq \frac{\text{MIN_NCC} \cdot (\text{MIN_NCC} - 1)}{2} + \frac{(\text{NVERTEX} - \text{MIN_NCC}) \cdot (\text{NVERTEX} - \text{MIN_NCC} - 1)}{2} \quad (3.115)$$

$$\text{arc_gen} = \text{CLIQUE}(>) : \text{NARC} \leq \frac{\text{MIN_NCC} \cdot (\text{MIN_NCC} - 1)}{2} + \frac{(\text{NVERTEX} - \text{MIN_NCC}) \cdot (\text{NVERTEX} - \text{MIN_NCC} - 1)}{2} \quad (3.116)$$

$$\text{arc_gen} = \text{CLIQUE}(\neq) : \text{NARC} \leq \text{MIN_NCC}^2 - \text{MIN_NCC} + (\text{NVERTEX} - \text{MIN_NCC})^2 - (\text{NVERTEX} - \text{MIN_NCC}) \quad (3.117)$$

$$\text{arc_gen} = \text{CYCLE} : \text{NARC} \leq \text{NVERTEX} - 4 \cdot (\text{MIN_NCC} < \text{NVERTEX}) \quad (3.118)$$

$$\text{arc_gen} = \text{PATH} : \text{NARC} \leq \max(0, \text{MIN_NCC} - 1) + \max(0, \text{NVERTEX} - \text{MIN_NCC} - 1) \quad (3.119)$$

Proof. (3.110) The maximum number of vertices according to a fixed number of vertices **NVERTEX** and to the fact that there is a connected component with **MIN_NCC** vertices is obtained by:

- Building a connected component with **MIN_NCC** vertices and creating an arc between each pair of vertices.
- Building a connected component with all the **NVERTEX** – **MIN_NCC** remaining vertices and creating an arc between each pair of vertices.

□

Proposition 84.

$$\text{MIN_NCC} > 1 \Rightarrow$$

$$\text{NARC} \geq \left\lfloor \frac{\text{NVERTEX}}{\text{MIN_NCC}} \right\rfloor \cdot (\text{MIN_NCC} - 1) + \text{NVERTEX} \bmod \text{MIN_NCC} \quad (3.120)$$

Proof. Achieving the minimum number of arcs with a fixed number of vertices and with a minimum number of vertices greater than or equal to one in each connected component is achieved in the following way:

- Since the minimum number of arcs of a connected component of n vertices is $n - 1$, splitting a connected component into k parts that all have more than one vertex saves $k - 1$ arcs. Therefore we build a maximum number of connected components. Since each connected component has at least **MIN_NCC** vertices we get $\lfloor \frac{\text{NVERTEX}}{\text{MIN_NCC}} \rfloor$ connected components.
- Since we can't build a connected component with the rest of the vertices (i.e. **NVERTEX** mod **MIN_NCC** vertices left) we have to incorporate them in the previous connected components and this costs one arc for each vertex.

□

When **MIN_NCC** = 1, note that Proposition 50 provides a lower bound on the number of arcs.

$$\boxed{\text{MIN_NCC}, \text{NCC}, \text{NVERTEX}}$$

Proposition 85.

$$\text{NVERTEX} \geq \text{NCC} \cdot \text{MIN_NCC} \quad (3.121)$$

Proof. The smallest number of vertices is obtained by taking all connected components to their minimum number of vertices **MIN_NCC**. □

MIN_NSCC, NARC, NVERTEX

Proposition 86.

$$\text{NARC} \leq \text{NVERTEX}^2 + \text{MIN_NSCC}^2 - \text{NVERTEX} \cdot \text{MIN_NSCC} \quad (3.122)$$

Proof. Achieving the maximum number of arcs, provided that we have at least one strongly connected component with MIN_NSCC vertices, is done by:

- Building a first strongly connected component \mathcal{C}_1 with MIN_NSCC vertices and adding an arc between each pair of vertices of \mathcal{C}_1 .
- Building a second strongly connected component \mathcal{C}_2 with $\text{NVERTEX} - \text{MIN_NSCC}$ vertices and adding an arc between each pair of vertices of \mathcal{C}_2 .

Finally, we add an arc from every vertex of \mathcal{C}_1 to every vertex of \mathcal{C}_2 . This leads to a total number of arcs of $\text{MIN_NSCC}^2 + (\text{NVERTEX} - \text{MIN_NSCC})^2 + \text{MIN_NSCC} \cdot (\text{NVERTEX} - \text{MIN_NSCC})$. \square

MIN_NSCC, NSCC, NVERTEX

Proposition 87.

$$\text{NVERTEX} \geq \text{NSCC} \cdot \text{MIN_NSCC} \quad (3.123)$$

Proof. Since each strongly connected component contains at least MIN_NSCC vertices the total number of vertices is greater than or equal to $\text{NSCC} \cdot \text{MIN_NSCC}$. \square

NARC, NCC, NVERTEX

Proposition 88.

$$\text{NARC} \leq (\text{NVERTEX} - \text{NCC} + 1)^2 + \text{NCC} - 1 \quad (3.124)$$

$$\text{arc_gen} = \text{CIRCUIT} : \text{NARC} \leq \text{NVERTEX} - \text{NCC} + 1 - (\text{NCC} \neq 1) \quad (3.125)$$

$$\text{arc_gen} = \text{CHAIN} : \text{NARC} \leq 2 \cdot \text{NVERTEX} - 2 \cdot \text{NCC} \quad (3.126)$$

$$\text{arc_gen} = \text{CLIQUE}(\leq) : \text{NARC} \leq \text{NCC} - 1 + \frac{(\text{NVERTEX} - \text{NCC} + 1) \cdot (\text{NVERTEX} - \text{NCC} + 2)}{2} \quad (3.127)$$

$$\text{arc_gen} = \text{CLIQUE}(\geq) : \text{NARC} \leq \text{NCC} - 1 + \frac{(\text{NVERTEX} - \text{NCC} + 1) \cdot (\text{NVERTEX} - \text{NCC} + 2)}{2} \quad (3.128)$$

$$\text{arc_gen} = \text{CLIQUE}(<) : \text{NARC} \leq \text{NCC} - 1 + \frac{(\text{NVERTEX} - \text{NCC} + 1) \cdot (\text{NVERTEX} - \text{NCC})}{2} \quad (3.129)$$

$$\text{arc_gen} = \text{CLIQUE}(>) : \text{NARC} \leq \text{NCC} - 1 + \frac{(\text{NVERTEX} - \text{NCC} + 1) \cdot (\text{NVERTEX} - \text{NCC})}{2} \quad (3.130)$$

$$\begin{aligned} \mathbf{arc_gen} = \mathit{CLIQUE}(\neq) : \mathbf{NARC} \leq \max(0, \mathbf{NCC} - 1) + \\ (\mathbf{NVERTEX} - \mathbf{NCC} + 1)^2 - (\mathbf{NVERTEX} - \mathbf{NCC} + 1) \end{aligned} \quad (3.131)$$

$$\mathbf{arc_gen} = \mathit{CYCLE} : \mathbf{NARC} \leq 2 \cdot \mathbf{NVERTEX} - 2 \cdot \mathbf{NCC} + 2 \cdot (\mathbf{NCC} = 1) \quad (3.132)$$

$$\mathbf{arc_gen} = \mathit{PATH} : \mathbf{NARC} = \mathbf{NVERTEX} - \mathbf{NCC} \quad (3.133)$$

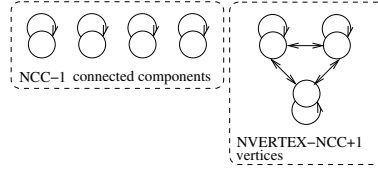


Figure 3.2: Illustration of Proposition 88. A graph that achieves the maximum number of arcs according to a fixed number of connected components as well as to a fixed number of vertices ($\mathbf{NCC} = 5, \mathbf{NVERTEX} = 7, \mathbf{NARC} = (7 - 5 + 1)^2 + 5 - 1 = 13$)

Proof. (3.124) We proceed by induction on $T(G) = \mathbf{NVERTEX}(G) - |X| - (\mathbf{NCC}(G) - 1)$, where X is any connected component of G of maximum cardinality. For $T(G) = 0$ then either $\mathbf{NCC}(G) = 1$ and thus the formula is clearly true, or all the connected components of G , but possibly X , are reduced to one element. Since isolated vertices are not allowed, the formula holds.

Assume that $T(G) \geq 1$. Then there exists Y , a connected component of G distinct from X , with more than one vertex. Let $y \in Y$ and let G' be the graph such that $V(G') = V(G)$ and $E(G')$ is defined by:

- For all Z connected components of G distinct from X and Y we have $G'[Z] = G[Z]$.
- With $X' = X \cup \{y\}$ and $Y' = Y - \{y\}$, we have $G'[Y'] = G[Y']$ and $E(G'[X']) = E(G[X]) \cup \bigcup_{x \in X'} \{(x, y), (y, x)\}$.

Clearly $|E(G')| - |E(G)| \geq 2 \cdot |X| + 1 - (2 \cdot |Y| - 1)$ and since X is of maximal cardinality the difference is strictly positive. Now as $\mathbf{NVERTEX}(G') = \mathbf{NVERTEX}(G)$, $\mathbf{NCC}(G') = \mathbf{NCC}(G)$ and as $T(G') = T(G) - 1$ the result holds by induction hypothesis. \square

Proposition 89.

$$\mathbf{NARC} \geq \mathbf{NVERTEX} - \mathbf{NCC} \quad (3.134)$$

equivalence : $\mathbf{NCC} > 0 \Rightarrow$

$$\begin{aligned} \mathbf{NARC} \geq (\mathbf{NVERTEX} \bmod \mathbf{NCC}) \cdot \left(\left\lfloor \frac{\mathbf{NVERTEX}}{\mathbf{NCC}} \right\rfloor + 1 \right)^2 + \\ (\mathbf{NCC} - \mathbf{NVERTEX} \bmod \mathbf{NCC}) \cdot \left\lfloor \frac{\mathbf{NVERTEX}}{\mathbf{NCC}} \right\rfloor^2 \end{aligned} \quad (3.135)$$

Proof. (3.134) By induction of the number of vertices. The formula holds for one vertex. Let G a graph with $n + 1$ vertices ($n \geq 1$). First assume there exists x in G such that $G - x$ has the same number of connected components than G . Since $\mathbf{NARC}(G) \geq \mathbf{NARC}(G - x) + 1$, and by induction hypothesis $\mathbf{NARC}(G - x) \geq \mathbf{NVERTEX}(G - x) - \mathbf{NCC}(G - x)$ the result holds. Otherwise all connected components of G are reduced to one vertex and the formula holds. \square

NARC, NSCC, NVERTEX

Proposition 90.

$$\mathbf{NARC} \leq (\mathbf{NVERTEX} - \mathbf{NSCC} + 1) \cdot \mathbf{NVERTEX} + \frac{\mathbf{NSCC} \cdot (\mathbf{NSCC} - 1)}{2} \quad (3.136)$$

$$\text{equivalence : } \mathbf{NARC} \leq \mathbf{NSCC} - 1 + (\mathbf{NVERTEX} - \mathbf{NSCC} + 1)^2 \quad (3.137)$$

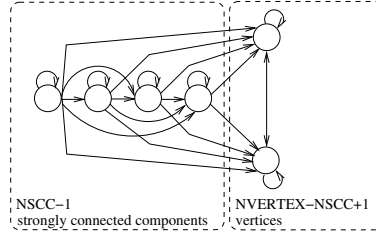


Figure 3.3: Illustration of Proposition 90(3.136). A graph that achieves the maximum number of arcs according to a fixed number of strongly connected components as well as to a fixed number of vertices ($\mathbf{NSCC} = 5$, $\mathbf{NVERTEX} = 6$, $\mathbf{NARC} = (6 - 5 + 1) \cdot 6 + \frac{5 \cdot (5 - 1)}{2} = 22$)

Proof. For proving 3.136, it is easier to rewrite the formula as $\mathbf{NARC} \leq (\mathbf{NVERTEX} - (\mathbf{NSCC} - 1))^2 + (\mathbf{NSCC} - 1) \cdot (\mathbf{NVERTEX} - (\mathbf{NSCC} - 1)) + \frac{\mathbf{NSCC} \cdot (\mathbf{NSCC} - 1)}{2}$. We proceed by induction on $T(G) = \mathbf{NVERTEX}(G) - |X| - (\mathbf{NSCC}(G) - 1)$, where X is any strongly connected component of G of maximum cardinality.

For $T(G) = 0$ then either $\mathbf{NSCC}(G) = 1$ and thus the formula is clearly true, or all the strongly connected components of G , but possibly X , are reduced to one element. Since the maximum number of arcs in a directed acyclic graph of n vertices is $\frac{n \cdot (n - 1)}{2}$, and as the subgraph of G induced by all the strongly connected components of G excepted X is acyclic, the formula clearly holds.

Assume that $T(G) \geq 1$, let $(X_i)_{i \in I}$ be the family of strongly connected components of G , and let G_r be the reduced graph of G induced by $(X_i)_{i \in I}$ (that is $V(G_r) = I$ and $\forall i_1, i_2 \in I$, $(i_1, i_2) \in E(G_r)$ iff $\exists x_1 \in X_{i_1}, \exists x_2 \in X_{i_2}$ such that $(x_1, x_2) \in E$). Consider G' such that $V(G') = V(G)$ and $E(G')$ is defined by:

- For all strongly connected components Z of G we have $G'[Z] = G[Z]$.
- For σ be any topological sort of G_r , $\forall x_i \in X_i, \forall x_j \in X_j$, $(x_i, x_j) \in E(G')$ whenever i is less than j with respect to σ .

Notice that G' satisfies the following properties: $T(G') = T(G)$, $V(G') = V(G)$, $\mathbf{NSCC}(G') = \mathbf{NSCC}(G)$, $E(G) \subseteq E(G')$, $(X_i)_{i \in I}$ is still the family of strongly connected components of G' , and moreover, for every $i \in I$ and every $x_i \in X_i$ we have that x_i is connected to any vertex outside X_i , that is the number of arcs incident to x_i and incident to vertices outside X_i is exactly $|V(G')| - |X_i|$.

Now, as $T(G') \geq 1$, there exists Y , a strongly connected component of G' distinct from X , with more than one vertex. Let $y \in Y$ and let G'' be the graph such that $V(G'') = V(G')$ and $E(G'')$ is defined by:

- $G''[V(G) - \{y\}] = G'[V(G) - \{y\}]$.

- With $X' = X \cup \{y\}$, we have $G''[Y'] = G'[Y']$ and $E(G''[X']) = E(G'[X]) \cup (\bigcup_{x \in X'} \{(x, y), (y, x)\})$.
- Assume that $X = X_j$ for $j \in I$. Then $\forall i \in I - \{j\}, \forall x_i \in X_i, (x_i, y) \in E(G'')$ whenever i is less than j with respect to σ and $(y, x_i) \in E(G'')$ whenever j is less than i with respect to σ .

Clearly $|E(G'')| - |E(G')| \geq 2|X| + 1 + |V(G')| - |X| - (2 \cdot |Y| - 1 + |V(G')| - |Y|) = |X| - |Y| + 2$ and since X is of maximal cardinality the difference is strictly positive. As $E(G) \subseteq E(G')$, $|E(G'')| - |E(G)|$ is also strictly positive. Now as $\mathbf{NVERTEX}(G'') = \mathbf{NVERTEX}(G') = \mathbf{NVERTEX}(G)$, $\mathbf{NSCC}(G'') = \mathbf{NSCC}(G') = \mathbf{NSCC}(G)$ and as $T(G'') = T(G') - 1 = T(G) - 1$ the result holds by induction hypothesis. \square

Proposition 91.

$$\mathbf{NARC} \geq \mathbf{NVERTEX} - \left\lfloor \frac{\mathbf{NSCC} - 1}{2} \right\rfloor \quad (3.138)$$

equivalence : $\mathbf{NSCC} > 0 \Rightarrow$

$$\begin{aligned} \mathbf{NARC} \geq (\mathbf{NVERTEX} \bmod \mathbf{NSCC}) \cdot \left(\left\lfloor \frac{\mathbf{NVERTEX}}{\mathbf{NSCC}} \right\rfloor + 1 \right)^2 + \\ (\mathbf{NSCC} - \mathbf{NVERTEX} \bmod \mathbf{NSCC}) \cdot \left\lfloor \frac{\mathbf{NVERTEX}}{\mathbf{NSCC}} \right\rfloor^2 \end{aligned} \quad (3.139)$$

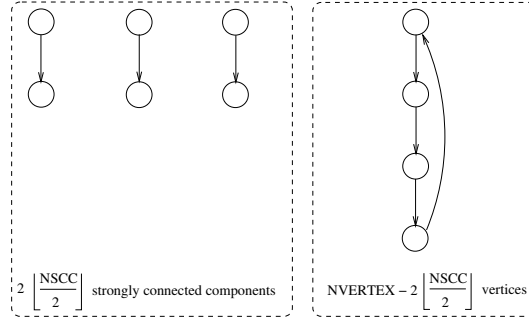


Figure 3.4: Illustration of Proposition 3.138. A graph that achieves the minimum number of arcs according to a fixed number of strongly connected components as well as to a fixed number of vertices ($\mathbf{NSCC} = 7, \mathbf{NVERTEX} = 10, \mathbf{NARC} = 10 - \lfloor \frac{7}{2} \rfloor = 7$)

Proof. For proving part 3.138 of Proposition 91 we proceed by induction on $\mathbf{NSCC}(G)$. If $\mathbf{NSCC}(G) = 1$ then, we have $\mathbf{NARC}(G) \geq \mathbf{NVERTEX}(G)$ (i.e. for one vertex this is true since every vertex has at least one arc, otherwise every vertex v has an arc arriving on v as well as an arc starting from v , thus we have $\mathbf{NARC} \geq \frac{2 \cdot \mathbf{NVERTEX}}{2}$). If $\mathbf{NSCC}(G) > 1$ let X be a strongly connected component of G . Then $\mathbf{NARC}(G) \geq \mathbf{NARC}(G[V(G) - X]) + \mathbf{NARC}(G[X])$. By induction hypothesis $\mathbf{NARC}(G[V(G) - X]) \geq |V(G) - X| - \left\lfloor \frac{\mathbf{NSCC}(G[V(G) - X]) - 1}{2} \right\rfloor$, thus $\mathbf{NARC}(G[V(G) - X]) \geq |V(G) - X| - \left\lfloor \frac{(\mathbf{NSCC}(G) - 1) - 1}{2} \right\rfloor$. Since $\mathbf{NARC}(G[X]) \geq |X|$ we obtain $\mathbf{NARC}(G) \geq |V(G)| - \left\lfloor \frac{(\mathbf{NSCC}(G) - 1) - 1}{2} \right\rfloor$, and thus the result holds. \square

Proposition 92.

$$\text{equivalence : } \mathbf{NVERTEX} > 0 \Rightarrow \mathbf{NSCC} \geq \left\lceil \frac{\mathbf{NVERTEX}^2}{\mathbf{NARC}} \right\rceil \quad (3.140)$$

Proof. As shown in [54], a lower bound for the minimum number of equivalence classes (e.g. strongly connected components) is the independence number of the graph and the right-hand side of Proposition 92 corresponds to a lower bound of the independence number proposed by Turán [55]. \square

NARC, NSINK, NVERTEX
Proposition 93.

$$\mathbf{NARC} \leq (\mathbf{NVERTEX} - \mathbf{NSINK}) \cdot \mathbf{NVERTEX} \quad (3.141)$$

Proof. The maximum number of arcs is achieved by the following pattern: For all non-sink vertices we have an arc to all vertices. \square

Proposition 94.

$$\mathbf{NARC} \geq \mathbf{NSINK} + \max(0, \mathbf{NVERTEX} - 2 \cdot \mathbf{NSINK}) \quad (3.142)$$

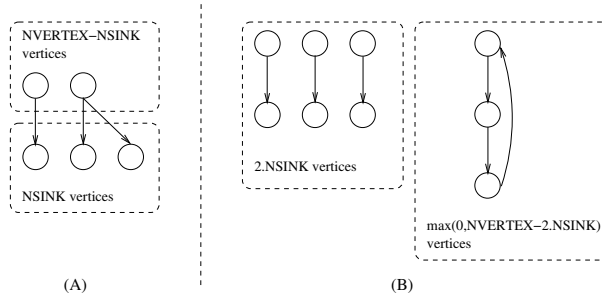


Figure 3.5: Illustration of Proposition 94. Graphs that achieve the minimum number of arcs according to a fixed number of sinks as well as to a fixed number of vertices (A : $\mathbf{NSINK} = 3, \mathbf{NVERTEX} = 5, \mathbf{NARC} = 3 + \max(0, 5 - 2 \cdot 3) = 3$; B : $\mathbf{NSINK} = 3, \mathbf{NVERTEX} = 9, \mathbf{NARC} = 3 + \max(0, 9 - 2 \cdot 3) = 6$)

Proof. Recall that for $x \in V(G)$, we have that $d_G^+(x) + d_G^-(x) \geq 1$. If x is a sink then $d_G^-(x) \geq 1$, consequently $\mathbf{NARC}(G) \geq \mathbf{NSINK}(G)$. If x is not a sink then $d_G^+(x) \geq 1$, consequently $\mathbf{NARC}(G) \geq |V(G)| - \mathbf{NSINK}(G)$. \square

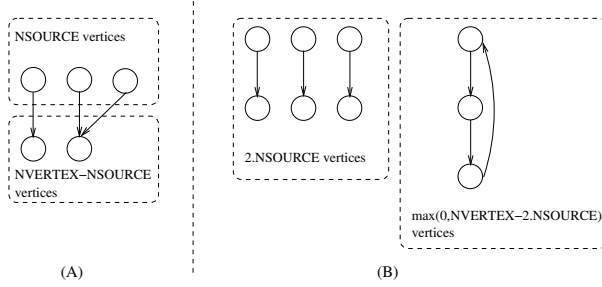


Figure 3.6: Illustration of Proposition 96. Graphs that achieve the minimum number of arcs according to a fixed number of sources as well as to a fixed number of vertices ($A : \mathbf{NSOURCE} = 3, \mathbf{NVERTEX} = 5, \mathbf{NARC} = 3 + \max(0, 5 - 2 \cdot 3) = 3$; $B : \mathbf{NSOURCE} = 3, \mathbf{NVERTEX} = 9, \mathbf{NARC} = 3 + \max(0, 9 - 2 \cdot 3) = 6$)

NARC, NSOURCE, NVERTEX

Proposition 95.

$$\mathbf{NARC} \leq (\mathbf{NVERTEX} - \mathbf{NSOURCE}) \cdot \mathbf{NVERTEX} \quad (3.143)$$

Proof. The maximum number of arcs is achieved by the following pattern: For all non-source vertices we have an arc from all vertices. \square

Proposition 96.

$$\mathbf{NARC} \geq \mathbf{NSOURCE} + \max(0, \mathbf{NVERTEX} - 2 \cdot \mathbf{NSOURCE}) \quad (3.144)$$

Proof. Similar to Proposition 94. \square

NSINK, NSOURCE, NVERTEX

Proposition 97.

$$\mathbf{NVERTEX} \geq \mathbf{NSOURCE} + \mathbf{NSINK} \quad (3.145)$$

Proof. No vertex can be both a source and a sink (isolated vertices are removed). \square

Graph invariants involving four characteristics of a final graph

MAX_NCC, MIN_NCC, NARC, NCC

Proposition 98. Let α denote $\max(0, \text{NCC} - 1)$.

$$\text{NARC} \leq \alpha \cdot \text{MAX_NCC}^2 + \text{MIN_NCC}^2 \quad (3.146)$$

$$\text{arc_gen} = \text{CIRCUIT} : \text{NARC} \leq \alpha \cdot \text{MAX_NCC} + \text{MIN_NCC} \quad (3.147)$$

$$\text{arc_gen} = \text{CHAIN} : \text{NARC} \leq \alpha \cdot (2 \cdot \text{MAX_NCC} - 2) + 2 \cdot \text{MIN_NCC} - 2 \quad (3.148)$$

$$\text{arc_gen} \in \{ \text{CLIQUE}(\leq), \text{CLIQUE}(\geq) \} : \text{NARC} \leq \alpha \cdot \frac{\text{MAX_NCC} \cdot (\text{MAX_NCC} + 1)}{2} + \frac{\text{MIN_NCC} \cdot (\text{MIN_NCC} + 1)}{2} \quad (3.149)$$

$$\text{arc_gen} \in \{ \text{CLIQUE}(<), \text{CLIQUE}(>) \} : \text{NARC} \leq \alpha \cdot \frac{\text{MAX_NCC} \cdot (\text{MAX_NCC} - 1)}{2} + \frac{\text{MIN_NCC} \cdot (\text{MIN_NCC} - 1)}{2} \quad (3.150)$$

$$\text{arc_gen} = \text{CLIQUE}(\neq) : \text{NARC} \leq \text{MIN_NCC}^2 - \text{MIN_NCC} + \alpha \cdot (\text{MAX_NCC}^2 - \text{MAX_NCC}) \quad (3.151)$$

$$\text{arc_gen} = \text{CYCLE} : \text{NARC} \leq 2 \cdot \alpha \cdot \text{MAX_NCC} + 2 \cdot \text{MIN_NCC} \quad (3.152)$$

$$\text{arc_gen} = \text{PATH} : \text{NARC} \leq \alpha \cdot (\text{MAX_NCC} - 1) + \text{MIN_NCC} - 1 \quad (3.153)$$

Proof. We construct $\text{NCC} - 1$ connected components with MAX_NCC vertices and one connected component with MIN_NCC vertices. n^2 corresponds to the maximum number of arcs in a connected component. $n, 2 \cdot n - 2, \frac{n \cdot (n+1)}{2}, \frac{n \cdot (n+1)}{2}, \frac{n \cdot (n-1)}{2}, \frac{n \cdot (n-1)}{2}, n^2 - n, 2 \cdot n$ and $n - 1$ respectively correspond to the maximum number of arcs in a connected component of n vertices according to the fact that we use the arc generator *CIRCUIT*, *CHAIN*, *CLIQUE*(\leq), *CLIQUE*(\geq), *CLIQUE*($<$), *CLIQUE*($>$), *CLIQUE*(\neq), *CYCLE* or *PATH*. \square

Proposition 99.

$$\text{NCC} > 0 \Rightarrow \text{NARC} \geq (\text{NCC} - 1) \cdot \max(1, \text{MIN_NCC} - 1) + \max(1, \text{MAX_NCC} - 1) \quad (3.154)$$

$$\text{arc_gen} = \text{PATH} : \text{NARC} \geq \max(0, \text{NCC} - 1) \cdot (\text{MIN_NCC} - 1) + \text{MAX_NCC} - 1 \quad (3.155)$$

Proof. (3.154) We construct $\text{NCC} - 1$ connected components with MIN_NCC vertices and one connected component with MAX_NCC vertices. The quantity $\max(1, n - 1)$ corresponds to the minimum number of arcs in a connected component of n ($n > 0$) vertices. \square

MAX_NCC, MIN_NCC, NCC, NVERTEX

Proposition 100.

$$\text{NVERTEX} \leq \max(0, \text{NCC} - 1) \cdot \text{MAX_NCC} + \text{MIN_NCC} \quad (3.156)$$

Proof. Derived from the definitions of MIN_NCC and MAX_NCC. □

Proposition 101.

$$\text{NVERTEX} \geq \max(0, \text{NCC} - 1) \cdot \text{MIN_NCC} + \text{MAX_NCC} \quad (3.157)$$

Proof. Derived from the definitions of MIN_NCC and MAX_NCC. □

MAX_NSICC, MIN_NSICC, NARC, NSICC

Proposition 102.

$$\begin{aligned} \text{NARC} \leq & \max(0, \text{NSICC} - 1) \cdot \text{MAX_NSICC}^2 + \text{MIN_NSICC}^2 + \\ & \max(0, \text{NSICC} - 1) \cdot \text{MIN_NSICC} \cdot \text{MAX_NSICC} + \\ & \text{MAX_NSICC}^2 \cdot \frac{\max(0, \text{NSICC} - 2) \cdot \max(0, \text{NSICC} - 1)}{2} \end{aligned} \quad (3.158)$$

Proof. We assume that we have at least two strongly connected components (the case with one being obvious). Let $(SCC_i)_{i \in [\text{NCC}(G)]}$ be the family of strongly connected components of G . Then $|E(G)| \leq \sum_{i \in [\text{NCC}(G)]} |E(G[SCC_i])| + k$, where k is the number of arcs between the distinct strongly connected components of G . For any strongly connected component SCC_i the number of arcs it has with the other strongly connected components is bounded by $|SCC_i| \cdot (|V(G) - SCC_i|)$. Consequently, $k \leq \frac{1}{2} \cdot \sum_{i \in [\text{NCC}(G)]} |SCC_i| \cdot (|V(G) - SCC_i|)$. W.l.o.g. we assume $|SCC_1| = \text{MIN_NCC}$. Then we get $k \leq \frac{1}{2} \cdot (\text{MIN_NCC} \cdot (\text{NCC} - 1) \cdot \text{MAX_NCC} + \text{MAX_NCC} \cdot ((\text{NCC} - 2) \cdot \text{MAX_NCC} + \text{MIN_NCC}))$. □

Proposition 103.

$$\text{NARC} \geq \max(0, \text{NSICC} - 1) \cdot \text{MIN_NSICC} + \text{MAX_NSICC} \quad (3.159)$$

Proof. Let $(SCC_i)_{i \in [\text{NCC}(G)]}$ be the family of strongly connected components of G , as $|E(G)| \geq \sum_{i \in [\text{NCC}(G)]} |E(G[SCC_i])|$, we obtain the result since in a strongly connected graph the number of edges is at least its number of vertices. □

MAX_NSICC, MIN_NSICC, NSICC, NVERTEX

Proposition 104.

$$\text{NVERTEX} \leq \max(0, \text{NSICC} - 1) \cdot \text{MAX_NSICC} + \text{MIN_NSICC} \quad (3.160)$$

Proof. Derived from the definitions of MIN_NSICC and MAX_NSICC. □

Proposition 105.

$$\text{NVERTEX} \geq \max(0, \text{NSICC} - 1) \cdot \text{MIN_NSICC} + \text{MAX_NSICC} \quad (3.161)$$

Proof. Derived from the definitions of MIN_NSICC and MAX_NSICC. □

MIN_NCC, NARC, NCC, NVERTEX

Proposition 106. Let α , β and γ respectively denote $\max(0, \text{NCC} - 1)$, $\text{NVERTEX} - \alpha \cdot \text{MIN_NCC}$ and MIN_NCC .

$$\text{NARC} \leq \alpha \cdot \gamma^2 + \beta^2 \quad (3.162)$$

$$\text{arc_gen} \in \{\text{CLIQUE}(\leq), \text{CLIQUE}(\geq)\} : \text{NARC} \leq \alpha \cdot \frac{\gamma \cdot (\gamma + 1)}{2} + \frac{\beta \cdot (\beta + 1)}{2} \quad (3.163)$$

$$\text{arc_gen} \in \{\text{CLIQUE}(<), \text{CLIQUE}(>)\} : \text{NARC} \leq \alpha \cdot \frac{\gamma \cdot (\gamma - 1)}{2} + \frac{\beta \cdot (\beta - 1)}{2} \quad (3.164)$$

$$\text{arc_gen} = \text{CLIQUE}(\neq) : \text{NARC} \leq \alpha \cdot \gamma \cdot (\gamma - 1) + \beta \cdot (\beta - 1) \quad (3.165)$$

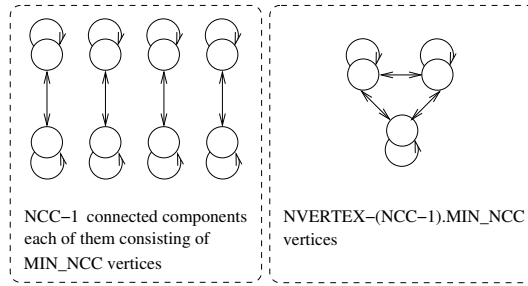


Figure 3.7: Illustration of Proposition 106(3.162). Graphs that achieve the maximum number of arcs according to a minimum number of vertices in a connected component, to a number of connected components, as well as to a fixed number of vertices ($\text{MIN_NCC} = 2$, $\text{NCC} = 5$, $\text{NVERTEX} = 11$, $\text{NARC} = (11 - (5 - 1) \cdot 2)^2 + (5 - 1) \cdot 2^2 = 25$)

Proof. For proving inequality 3.162 we proceed by induction on the number of vertices of G . First note that if all the connected components are reduced to one element the result is obvious. Thus we assume that the number of vertices in the maximal sized connected component of G is at least 2. Let x be an element of the maximal sized connected component of G . Then, $G - x$ satisfies $\alpha(G - x) = \alpha(G)$, $\gamma(G - x) = \gamma(G)$ and $\beta(G - x) = \beta(G) - 1$. Since by induction hypothesis $|E(G - x)| \leq \alpha(G - x) \cdot \gamma(G - x)^2 + \beta(G - x)^2$, and since the number of arcs of G incident to x is at most $2 \cdot (\beta(G) - 1) + 1$, we have that $|E(G)| \leq \alpha(G) \cdot \gamma(G)^2 + (\beta(G) - 1)^2 + 2 \cdot (\beta(G) - 1) + 1$. And thus the result follows. \square

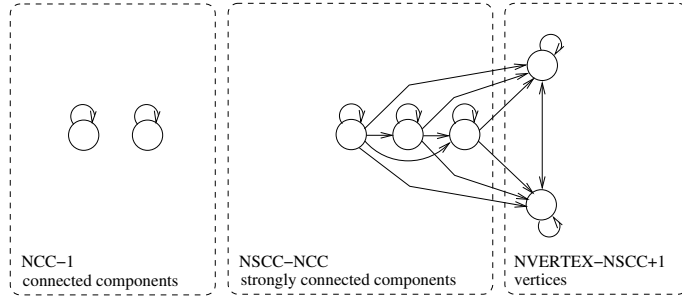


Figure 3.8: Illustration of Proposition 107. A graph that achieves the maximum number of arcs according to a fixed number of connected components, to a fixed number of strongly connected components as well as to a fixed number of vertices ($\mathbf{NCC} = 3$, $\mathbf{NSCC} = 6$, $\mathbf{NVERTEX} = 7$, $\mathbf{NARC} = 3 - 1 + (7 - 6 + 1)(7 - 3 + 1) + (6 - 3 + 1)(6 - 3)/2 = 18$)

NARC, NCC, NSCC, NVERTEX

Proposition 107.

$$\mathbf{NARC} \leq \mathbf{NCC} - 1 + (\mathbf{NVERTEX} - \mathbf{NSCC} + 1) \cdot (\mathbf{NVERTEX} - \mathbf{NCC} + 1) + \frac{(\mathbf{NSCC} - \mathbf{NCC} + 1) \cdot (\mathbf{NSCC} - \mathbf{NCC})}{2} \quad (3.166)$$

Proof. We proceed by induction on $T(G) = \mathbf{NVERTEX}(G) - |X| - (\mathbf{NCC}(G) - 1)$, where X is any connected component of G of maximum cardinality. For $T(G) = 0$ then either $\mathbf{NCC}(G) = 1$ and thus the formula is clearly true, by Proposition 3.136 or all the connected components of G , but possibly X , are reduced to one element. Since isolated vertices are not allowed, again by Proposition 3.136 applied on $G[X]$, the formula holds indeed $\mathbf{NVERTEX}(G[X]) = \mathbf{NVERTEX}(G) - (\mathbf{NCC}(G) - 1)$ and $\mathbf{NSCC}(G[X]) = \mathbf{NSCC}(G) - (\mathbf{NCC}(G) - 1)$.

Assume that $T(G) \geq 1$. Then there exists Y , a connected component of G distinct from X , with more than one vertex.

- Firstly assume that $G[Y]$ is strongly connected. Let $y \in Y$ and let G' be the graph such that $V(G') = V(G)$ and $E(G')$ is defined by:
 - For all Z connected components of G distinct from X and Y we have $G'[Z] = G[Z]$.
 - With $X' = X \cup (Y - \{y\})$ and $Y' = \{y\}$, we have $E(G'[Y']) = \{(y, y)\}$, $E(G'[X']) = E(G[X]) \cup \{(z, x) : z \in Y - \{y\}, x \in X\} \cup \{(z, t) : z, t \in Y - \{y\}\}$.

Clearly we have that $|E(G')| - |E(G)| \geq (|Y| - 1) \cdot |X| - 2 \cdot (|Y| - 1)$ and since $|X| \geq |Y| \geq 2$, the difference is positive or null. Now as $\mathbf{NVERTEX}(G') = \mathbf{NVERTEX}(G)$, $\mathbf{NCC}(G') = \mathbf{NCC}(G)$, $\mathbf{NSCC}(G') = \mathbf{NSCC}(G)$ (since $G'[Y - \{y\}]$ is strongly connected because $E(G'[Y - \{y\}]) = \{(z, t) : z, t \in Y - \{y\}\}$ and since the reduced graph of the strongly connected components of $G'[X']$ is exactly the reduced graph of the strongly connected components of $G[X]$ to which a unique source has been added) and as $T(G') \leq T(G) - 1$, the result holds by induction hypothesis.

- Secondly assume that $G[Y]$ is not strongly connected. Let $Z \subset Y$ such that Z is a strongly connected component of $G[Y]$ corresponding to a source in the reduced graph of the strongly connected components of $G[Y]$. Let G' be the graph such that $V(G') = V(G)$ and $E(G')$ is defined by:
 - For all W connected components of G distinct from X and Y we have $G'[W] = G[W]$.
 - With $X' = X \cup Z$ and $Y' = Y - Z$, we have $E(G'[Y']) = E(G[Y'])$ if $|Y'| > 1$ and $E(G'[Y']) = \{(y, y)\}$ if $Y' = \{y\}$. $E(G'[X']) = E(G[X]) \cup \{(z, x) : z \in Z, x \in X\}$.

Clearly we have that $|E(G')| - |E(G)| \geq |Z| \cdot |X| - |Z| \cdot (|Y| - |Z|)$ and since $|X| > |Y| - |Z|$, the difference is strictly positive. Now as $\mathbf{NVERTEX}(G') = \mathbf{NVERTEX}(G)$, $\mathbf{NCC}(G') = \mathbf{NCC}(G)$, $\mathbf{NSCC}(G') = \mathbf{NSCC}(G)$ and as $T(G') \leq T(G) - 1$, the result holds by induction hypothesis. □

Proposition 108.

$$\mathbf{NARC} \geq \mathbf{NVERTEX} - \max(0, \min(\mathbf{NCC}, \mathbf{NSCC} - \mathbf{NCC})) \quad (3.167)$$

Proof. We prove that the invariant is valid for any digraph G . First notice that for an operational behavior, since we can't assume that Proposition 53 (i.e. $\mathbf{NCC}(G) \leq \mathbf{NSCC}(G)$) was already triggered, we use the max operator. But since any strongly connected component is connected, then $\mathbf{NSCC}(G) - \mathbf{NCC}(G)$ is never negative. Consequently we only show by induction on $\mathbf{NSCC}(G)$ that $\mathbf{NARC}(G) \geq \mathbf{NVERTEX}(G) - \min(\mathbf{NCC}(G), \mathbf{NSCC}(G) - \mathbf{NCC}(G))$. To begin notice that if X is a strongly (non void) connected component then either $\mathbf{NARC}(G[X]) \geq |X|$ or $\mathbf{NARC}(G[X]) = 0$ and in this latter case we have that both $|X| = 1$ and X is strictly included in a connected component of G (recall that isolated vertices are not allowed). Thus we can directly assume that $\mathbf{NSCC}(G) = k > 1$.

First, consider that there exists a connected component of G , say X , which is also strongly connected. Let $G' = G - X$, consequently we have $\mathbf{NSCC}(G') = \mathbf{NSCC}(G) - 1$, $\mathbf{NCC}(G') = \mathbf{NCC}(G) - 1$, $\mathbf{NVERTEX}(G') = \mathbf{NVERTEX}(G) - |X|$, and $\mathbf{NARC}(G) \geq |X| + \mathbf{NARC}(G')$. Then $\mathbf{NARC}(G) \geq |X| + \mathbf{NVERTEX}(G') - \min(\mathbf{NCC}(G'), \mathbf{NSCC}(G') - \mathbf{NCC}(G'))$ and thus $\mathbf{NARC}(G) \geq \mathbf{NVERTEX}(G) - \min(\mathbf{NCC}(G) - 1, \mathbf{NSCC}(G) - \mathbf{NCC}(G))$, which immediately gives the result.

Second consider that any strongly connected component is strictly included in a connected component of G . Then, either there exists a strongly connected component X such that $|X| \geq 2$. Let $G' = G - X$, consequently we have $\mathbf{NSCC}(G') = \mathbf{NSCC}(G) - 1$, $\mathbf{NCC}(G') = \mathbf{NCC}(G)$, $\mathbf{NVERTEX}(G') = \mathbf{NVERTEX}(G) - |X|$, and $\mathbf{NARC}(G) \geq |X| + 1 + \mathbf{NARC}(G')$. Then $\mathbf{NARC}(G) \geq |X| + 1 + \mathbf{NVERTEX}(G') - \min(\mathbf{NCC}(G'), \mathbf{NSCC}(G') - \mathbf{NCC}(G'))$ and thus $\mathbf{NARC}(G) \geq \mathbf{NVERTEX}(G) + 1 - \min(\mathbf{NCC}(G), \mathbf{NSCC}(G) - \mathbf{NCC}(G) + 1)$, which immediately gives the result. Or, all the strongly connected components are reduced to one element, so we have $\mathbf{NSCC}(G) = \mathbf{NVERTEX}(G)$, and thus we obtain that $\mathbf{NVERTEX}(G) - \min(\mathbf{NCC}(G), \mathbf{NSCC}(G) - \mathbf{NCC}(G)) = \min(\mathbf{NCC}(G), \mathbf{NVERTEX}(G) - \mathbf{NCC}(G))$, which gives the result by for example Proposition 89 (3.134). □

This bound is tight: take for example any circuit.

NARC, NSINK, NSOURCE, NVERTEX

Proposition 109.

$$\begin{aligned} \text{NARC} \leq & \text{NVERTEX}^2 - \text{NVERTEX} \cdot \text{NSOURCE} \\ & - \text{NVERTEX} \cdot \text{NSINK} + \text{NSOURCE} \cdot \text{NSINK} \end{aligned} \quad (3.168)$$

Proof. Since the maximum number of arcs of a digraph is NVERTEX^2 , and since:

- No vertex can have a source as a successor we lose $\text{NVERTEX} \cdot \text{NSOURCE}$ arcs,
- No sink can have a successor we lose $\text{NVERTEX} \cdot \text{NSINK}$ arcs.

In these two sets of arcs we count twice the arcs from the sinks to the sources, so we finally get a maximum number of arcs corresponding to the right-hand side of the inequality to prove. \square

Graph invariants involving five characteristics of a final graph

MAX_NCC, MIN_NCC, NARC, NCC, NVERTEX

Proposition 110.

Let:

- $\Delta = \text{NVERTEX} - \text{NCC} \cdot \text{MIN_NCC}$,
- $\delta = \lfloor \frac{\Delta}{\max(1, \text{MAX_NCC} - \text{MIN_NCC})} \rfloor$,
- $r = \Delta \bmod \max(1, \text{MAX_NCC} - \text{MIN_NCC})$,
- $\epsilon = (r > 0)$.

$$\Delta = 0 \vee (\text{MAX_NCC} \neq \text{MIN_NCC} \wedge \delta + \epsilon \leq \text{NCC}) \quad (3.169)$$

$$\text{NARC} \leq (\text{NCC} - \delta - \epsilon) \cdot \text{MIN_NCC}^2 + \epsilon \cdot (\text{MIN_NCC} + r)^2 + \delta \cdot \text{MAX_NCC}^2 \quad (3.170)$$

Proposition 110 is currently a conjecture.

MIN_NCC, NARC, NCC, NSCC, NVERTEX

Proposition 111.

$$\begin{aligned} \text{NARC} \leq & (\text{NCC} - 1) \cdot \max(1, (\text{MIN_NCC} - 1)) + \\ & (\text{NVERTEX} - \text{NSCC} + 1) \cdot (\text{NVERTEX} - \text{NCC} + 1) + \\ & \frac{(\text{NSCC} - \text{NCC} + 1) \cdot (\text{NSCC} - \text{NCC})}{2} \end{aligned} \quad (3.171)$$

Proposition 111 is currently a conjecture.

Graph invariants relating two characteristics of two final graphs

$$\boxed{\text{MAX_NCC}_1, \text{NCC}_2}$$

Proposition 112.

$$\text{vpartition} : \text{MAX_NCC}_1 < \text{NVERTEX}_{\text{INITIAL}} \Leftrightarrow \text{NCC}_2 > 0 \quad (3.172)$$

$$\text{apartition} : \text{MAX_NCC}_1 < \text{NVERTEX}_{\text{INITIAL}} \Leftrightarrow \text{NCC}_2 > 0 \quad (3.173)$$

Proof. (3.172) Since we have the precondition vpartition , we know that each vertex of the initial graph belongs to the first or to the second final graphs (but not to both).

1. On the one hand, if the largest connected component of the first final graph can't contain all the vertices of the initial graph, then the second final graph has at least one connected component.
2. On the other hand, if the second final graph has at least one connected component then the largest connected component of the first final graph can't be equal to the initial graph.

(3.173) holds for a similar reason. \square

$$\boxed{\text{MAX_NCC}_2, \text{NCC}_1}$$

Proposition 113.

$$\text{vpartition} : \text{MAX_NCC}_2 < \text{NVERTEX}_{\text{INITIAL}} \Leftrightarrow \text{NCC}_1 > 0 \quad (3.174)$$

$$\text{apartition} : \text{MAX_NCC}_2 < \text{NVERTEX}_{\text{INITIAL}} \Leftrightarrow \text{NCC}_1 > 0 \quad (3.175)$$

Proof. Similar to Proposition 112. \square

$$\boxed{\text{MIN_NCC}_1, \text{NCC}_2}$$

Proposition 114.

$$\text{vpartition} : \text{MIN_NCC}_1 < \text{NVERTEX}_{\text{INITIAL}} \Leftrightarrow \text{NCC}_2 > 0 \quad (3.176)$$

Proof. Since we have the precondition vpartition , we know that each vertex of the initial graph belongs to the first or to the second final graphs (but not to both).

1. On the one hand, if the smallest connected component of the first final graph can't contain all the vertices of the initial graph, then the second final graph has at least one connected component.
2. On the other hand, if the second final graph has at least one connected component then the smallest connected component of the first final graph can't be equal to the initial graph.

\square

$\text{MIN_NCC}_2, \text{NCC}_1$

Proposition 115.

$$\text{vpartition} : \text{MIN_NCC}_2 < \text{NVERTEX}_{\text{INITIAL}} \Leftrightarrow \text{NCC}_1 > 0 \quad (3.177)$$

Proof. Similar to Proposition 114. □

$\text{NARC}_1, \text{NARC}_2$

Proposition 116.

$$\text{apartition} \wedge \text{arc_gen} = \text{PATH} : \text{NARC}_1 + \text{NARC}_2 = \text{NVERTEX}_{\text{INITIAL}} - 1 \quad (3.178)$$

Proof. Holds since each arc of the initial graph belongs to one of the two final graphs and since the initial graph has $\text{NVERTEX}_{\text{INITIAL}} - 1$ arcs. □

$\text{NCC}_1, \text{NCC}_2$

Proposition 117.

$$\text{apartition} \wedge \text{arc_gen} = \text{PATH} : |\text{NCC}_1 - \text{NCC}_2| \leq 1 \quad (3.179)$$

$$\text{vpartition} \wedge \text{consecutive_loops_are_connected} : |\text{NCC}_1 - \text{NCC}_2| \leq 1 \quad (3.180)$$

Proof. Holds because the two initial graphs correspond to a path and because consecutive connected components do not come from the same graph constraint. □

Proposition 118.

$$\text{apartition} \wedge \text{arc_gen} = \text{PATH} : \text{NCC}_1 + \text{NCC}_2 < \text{NVERTEX}_{\text{INITIAL}} \quad (3.181)$$

Proof. Holds because the initial graph is a path. □

$\text{NVERTEX}_1, \text{NVERTEX}_2$

Proposition 119.

$$\text{vpartition} : \text{NVERTEX}_1 + \text{NVERTEX}_2 = \text{NVERTEX}_{\text{INITIAL}} \quad (3.182)$$

Proof. By definition of vpartition each vertex of the initial graph belongs to one of the two final graphs (but not to both). □

Graph invariants relating three characteristics of two final graphs

$$\boxed{\text{MAX_NCC}_1, \text{MIN_NCC}_1, \text{MIN_NCC}_2}$$

Proposition 120.

$$\begin{aligned} & \text{apartition} \wedge \text{arc_gen} = \text{PATH} : \\ & \max(2, \text{MIN_NCC}_1) + \max(3, \text{MIN_NCC}_1 + 1, \text{MAX_NCC}_1) + \\ & \max(2, \text{MIN_NCC}_2) - 2 > \text{NVERTEX}_{\text{INITIAL}} \Rightarrow \text{MIN_NCC}_1 = \text{MAX_NCC}_1 \end{aligned} \quad (3.183)$$

Proof. The quantity $\max(2, \text{MIN_NCC}_1) + \max(3, \text{MIN_NCC}_1 + 1, \text{MAX_NCC}_1) + \max(2, \text{MIN_NCC}_2) - 2$ corresponds to the minimum number of variables needed for building two non-empty connected components of respective size MIN_NCC_1 and MAX_NCC_1 such that MAX_NCC_1 is strictly greater than MIN_NCC_1 . If this quantity is greater than the total number of variables we have that $\text{MIN_NCC}_1 = \text{MAX_NCC}_1$. \square

Proposition 121.

$$\begin{aligned} & \text{vpartition} \wedge \text{consecutive_loops_are_connected} : \\ & \max(1, \text{MIN_NCC}_1) + \max(2, \text{MIN_NCC}_1 + 1, \text{MAX_NCC}_1) + \\ & \max(1, \text{MIN_NCC}_2) > \text{NVERTEX}_{\text{INITIAL}} \Rightarrow \text{MIN_NCC}_1 = \text{MAX_NCC}_1 \end{aligned} \quad (3.184)$$

Proof. The quantity $\max(1, \text{MIN_NCC}_1) + \max(2, \text{MIN_NCC}_1 + 1, \text{MAX_NCC}_1) + \max(1, \text{MIN_NCC}_2)$ corresponds to the minimum number of variables needed for building two non-empty connected components of respective size MIN_NCC_1 and MAX_NCC_1 such that MAX_NCC_1 is strictly greater than MIN_NCC_1 . If this quantity is greater than the total number of variables we have that $\text{MIN_NCC}_1 = \text{MAX_NCC}_1$. \square

$$\boxed{\text{MAX_NCC}_2, \text{MIN_NCC}_2, \text{MIN_NCC}_1}$$

Proposition 122.

$$\begin{aligned} & \text{apartition} \wedge \text{arc_gen} = \text{PATH} : \\ & \max(2, \text{MIN_NCC}_2) + \max(3, \text{MIN_NCC}_2 + 1, \text{MAX_NCC}_2) + \\ & \max(2, \text{MIN_NCC}_1) - 2 > \text{NVERTEX}_{\text{INITIAL}} \Rightarrow \text{MIN_NCC}_2 = \text{MAX_NCC}_2 \end{aligned} \quad (3.185)$$

Proof. Similar to Proposition 120. \square

Proposition 123.

$$\begin{aligned} & \text{vpartition} \wedge \text{consecutive_loops_are_connected} : \\ & \max(1, \text{MIN_NCC}_2) + \max(2, \text{MIN_NCC}_2 + 1, \text{MAX_NCC}_2) + \\ & \max(1, \text{MIN_NCC}_1) > \text{NVERTEX}_{\text{INITIAL}} \Rightarrow \text{MIN_NCC}_2 = \text{MAX_NCC}_2 \end{aligned} \quad (3.186)$$

Proof. Similar to Proposition 121. \square

$\text{MIN_NCC}_1, \text{NARC}_2, \text{NCC}_1$
--

Proposition 124.

$$\begin{aligned} \text{apartition} \wedge \text{arc_gen} = \text{PATH} \wedge \text{NVERTEX}_{\text{INITIAL}} > 0 : \\ \text{NCC}_1 = 1 \Leftrightarrow \text{MIN_NCC}_1 + \text{NARC}_2 = \text{NVERTEX}_{\text{INITIAL}} \end{aligned} \quad (3.187)$$

Proof. When $\text{MIN_NCC}_1 + \text{NARC}_2 = \text{NVERTEX}_{\text{INITIAL}}$ there is no more room for an extra connected component for the first final graph. \square

$\text{MIN_NCC}_1, \text{NARC}_2, \text{NCC}_1$
--

Proposition 125.

$$\begin{aligned} \text{apartition} \wedge \text{arc_gen} = \text{PATH} \wedge \text{NVERTEX}_{\text{INITIAL}} > 0 : \\ \text{NCC}_2 = 1 \Leftrightarrow \text{MIN_NCC}_2 + \text{NARC}_1 = \text{NVERTEX}_{\text{INITIAL}} \end{aligned} \quad (3.188)$$

Proof. Similar to Proposition 124. \square

Graph invariants relating four characteristics of two final graphs

$\text{MAX_NCC}_1, \text{MIN_NCC}_1, \text{MIN_NCC}_2, \text{NCC}_1$

Proposition 126.

$$\begin{aligned} \text{apartition} \wedge \text{arc_gen} = \text{PATH} : \\ \max(2, \text{MIN_NCC}_1) + \max(2, \text{MAX_NCC}_1) + \max(2, \text{MIN_NCC}_2) - 2 > \\ \text{NVERTEX}_{\text{INITIAL}} \Rightarrow \text{NCC}_1 \leq 1 \end{aligned} \quad (3.189)$$

Proof. The quantity $\max(2, \text{MIN_NCC}_1) + \max(2, \text{MAX_NCC}_1) + \max(2, \text{MIN_NCC}_2) - 2$ corresponds to the minimum number of variables needed for building two non-empty connected components of respective size MIN_NCC_1 and MAX_NCC_1 . If this quantity is greater than the total number of variables we have that $\text{NCC}_1 \leq 1$. \square

Proposition 127.

$$\begin{aligned} \text{vpartition} \wedge \text{consecutive_loops_are_connected} : \\ \max(1, \text{MIN_NCC}_1) + \max(1, \text{MAX_NCC}_1) + \max(1, \text{MIN_NCC}_2) > \\ \text{NVERTEX}_{\text{INITIAL}} \Rightarrow \text{NCC}_1 \leq 1 \end{aligned} \quad (3.190)$$

Proof. The quantity $\max(1, \text{MIN_NCC}_1) + \max(1, \text{MAX_NCC}_1) + \max(1, \text{MIN_NCC}_2)$ corresponds to the minimum number of variables needed for building two non-empty connected components of respective size MIN_NCC_1 and MAX_NCC_1 . If this quantity is greater than the total number of variables we have that $\text{NCC}_1 \leq 1$. \square

$$\boxed{\text{MAX_NCC}_2, \text{MIN_NCC}_2, \text{MIN_NCC}_1, \text{NCC}_2}$$

Proposition 128.

$$\begin{aligned} & \text{apartition} \wedge \text{arc_gen} = \text{PATH} : \\ & \max(2, \text{MIN_NCC}_2) + \max(2, \text{MAX_NCC}_2) + \max(2, \text{MIN_NCC}_1) - 2 > \\ & \text{NVERTEX}_{\text{INITIAL}} \Rightarrow \text{NCC}_2 \leq 1 \end{aligned} \quad (3.191)$$

Proof. Similar to Proposition 126. \square

Proposition 129.

$$\begin{aligned} & \text{vpartition} \wedge \text{consecutive_loops_are_connected} : \\ & \max(1, \text{MIN_NCC}_2) + \max(1, \text{MAX_NCC}_2) + \max(1, \text{MIN_NCC}_1) > \\ & \text{NVERTEX}_{\text{INITIAL}} \Rightarrow \text{NCC}_2 \leq 1 \end{aligned} \quad (3.192)$$

Proof. Similar to Proposition 127. \square

Graph invariants relating five characteristics of two final graphs

$$\boxed{\text{MAX_NCC}_1, \text{MAX_NCC}_2, \text{MIN_NCC}_1, \text{MIN_NCC}_2, \text{NCC}_1}$$

Proposition 130.

$$\begin{aligned} & \text{vpartition} \wedge \text{consecutive_loops_are_connected} : \\ & \text{MIN_NCC}_1 \cdot \max(0, \text{NCC}_1 - 1) + \text{MAX_NCC}_1 + \\ & \text{MIN_NCC}_2 \cdot \max(0, \text{NCC}_1 - 2) + \text{MAX_NCC}_2 \leq \text{NVERTEX}_{\text{INITIAL}} \end{aligned} \quad (3.193)$$

Proof. The left-hand side of 130 corresponds to the minimum number of vertices of the two final graphs provided that we build the smallest possible connected components. \square

Proposition 131.

$$\begin{aligned} & \text{vpartition} \wedge \text{consecutive_loops_are_connected} : \\ & \overline{\text{NCC}_1} \leq (\overline{\text{MAX_NCC}_1} > 0) + \left\lfloor \frac{\alpha}{\beta} \right\rfloor + (\alpha \bmod \beta \geq \max(1, \underline{\text{MIN_NCC}_1})) \\ & \begin{cases} \bullet \alpha = \max(0, \text{NVERTEX}_{\text{INITIAL}} - \max(1, \underline{\text{MAX_NCC}_1}) - \max(1, \underline{\text{MAX_NCC}_2})), \\ \bullet \beta = \max(1, \underline{\text{MIN_NCC}_1}) + \max(1, \underline{\text{MIN_NCC}_2}). \end{cases} \end{aligned} \quad (3.194)$$

Proof. The maximum number of connected components is achieved by building non-empty groups as small as possible, except for two groups of respective size $\max(1, \underline{\text{MAX_NCC}_1})$ and $\max(1, \underline{\text{MAX_NCC}_2})$, which have to be built. \square

Proposition 132.

$$\begin{aligned} & \text{vpartition} \wedge \text{consecutive_loops_are_connected} : \\ & \text{MAX_NCC}_1 \cdot \max(0, \text{NCC}_1 - 1) + \text{MIN_NCC}_1 + \\ & \text{MAX_NCC}_2 \cdot \text{NCC}_1 + \text{MIN_NCC}_2 \geq \text{NVERTEX}_{\text{INITIAL}} \end{aligned} \quad (3.195)$$

Proof. The left-hand side of 132 corresponds to the maximum number of vertices of the two final graphs provided that we build the largest possible connected components. \square

Proposition 133.

$\text{vpartition} \wedge \text{consecutive_loops_are_connected} :$

$$\begin{aligned} \underline{\text{NCC}}_1 \geq (\overline{\text{MAX_NCC}}_2 < \text{NVERTEX}_{\text{INITIAL}}) + \left\lfloor \frac{\alpha}{\beta} \right\rfloor + (\alpha \bmod \beta > \overline{\text{MAX_NCC}}_2) \\ \left\{ \begin{array}{l} \bullet \alpha = \max(0, \text{NVERTEX}_{\text{INITIAL}} - \overline{\text{MIN_NCC}}_1 - \overline{\text{MIN_NCC}}_2, \\ \bullet \beta = \max(1, \overline{\text{MAX_NCC}}_1) + \max(1, \overline{\text{MAX_NCC}}_2). \end{array} \right. \end{aligned} \quad (3.196)$$

Proof. The minimum number of connected components is achieved by taking the groups as large as possible except for two groups of respective size $\overline{\text{MIN_NCC}}_2$ and $\overline{\text{MIN_NCC}}_1$, which have to be built. \square

Proposition 134.

$\text{vpartition} \wedge \text{consecutive_loops_are_connected} :$

$$\begin{aligned} \text{MAX_NCC}_2 \leq \max(\text{MIN_NCC}_2, \text{NVERTEX}_{\text{INITIAL}} - \alpha), \text{ with :} \\ \bullet \alpha = \text{MIN_NCC}_1 \cdot \max(0, \text{NCC}_1 - 1) + \text{MAX_NCC}_1 + \\ \text{MIN_NCC}_2 + \text{MIN_NCC}_2 \cdot \max(0, \text{NCC}_1 - 3) \end{aligned} \quad (3.197)$$

Proof. If $\text{NCC}_1 \leq 1$ we have that $\text{MAX_NCC}_2 \leq \text{MIN_NCC}_2$. Otherwise, when $\text{NCC}_1 > 1$, we have that $\text{MIN_NCC}_1 \cdot \max(0, \text{NCC}_1 - 1) + \text{MAX_NCC}_1 + \text{MIN_NCC}_2 + \text{MAX_NCC}_2 + \text{MIN_NCC}_2 \cdot \max(0, \text{NCC}_1 - 3) \leq \text{NVERTEX}_{\text{INITIAL}} - \text{NCC}_1 - 3$ comes from the fact that we build the minimum number of connected components in the second final graph (i.e. $\text{NCC}_1 - 1$ connected components) and that we have already built two connected components of respective size MIN_NCC_2 and MAX_NCC_2 . By isolating MAX_NCC_2 in the previous expression and by grouping the two inequalities the result follows. \square

Proposition 135.

$\text{apartition} \wedge \text{arc_gen} = \text{PATH} \wedge \text{MIN_NCC}_1 > 1 \wedge \text{MIN_NCC}_2 > 1 :$

$$\begin{aligned} \text{NCC}_1 \leq (\text{MAX_NCC}_1 > 0) + \left\lfloor \frac{\alpha}{\beta} \right\rfloor + ((\alpha \bmod \beta) + 1 \geq \text{MIN_NCC}_1), \text{ with :} \\ \left\{ \begin{array}{l} \bullet \alpha = \max(0, \text{NVERTEX}_{\text{INITIAL}} - \text{MAX_NCC}_1 - \text{MAX_NCC}_2 + 1), \\ \bullet \beta = \text{MIN_NCC}_1 + \text{MIN_NCC}_2 - 2. \end{array} \right. \end{aligned} \quad (3.198)$$

Proof. The maximum number of connected components of G_1 is achieved by:

- Building a first connected component of G_1 involving MAX_NCC_1 vertices,
- Building a first connected component of G_2 involving MAX_NCC_2 vertices,
- Building alternatively a connected component of G_1 and a connected component of G_2 involving respectively MIN_NCC_1 and MIN_NCC_2 vertices,
- Finally, if this is possible, building a connected component of G_1 involving MIN_NCC_1 vertices.

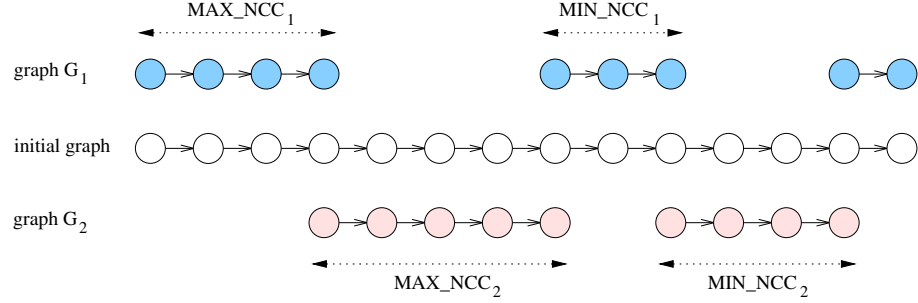


Figure 3.9: Illustration of Proposition 135. Configuration achieving the maximum number of connected components for G_1 according to the size of the smallest and largest connected components of G_1 and G_2 and to an initial number of vertices ($\text{MAX_NCC}_1 = 4$, $\text{MAX_NCC}_2 = 5$, $\text{MIN_NCC}_1 = 3$, $\text{MIN_NCC}_2 = 4$, $\text{NVERTEX_INITIAL} = 14$, $\alpha = \max(0, 14 - 4 - 5 + 1) = 6$, $\beta = \max(2, 3 + 4 - 2) = 5$, $\text{NCC}_1 = (4 > 0) + \lfloor \frac{6}{5} \rfloor + (((6 \bmod 5) + 1) \geq 3) = 2$)

□

Proposition 136.

$$\text{apartition} \wedge \text{arc_gen} = \text{PATH} \wedge \text{MIN_NCC}_1 > 1 \wedge \text{MIN_NCC}_2 > 1 :$$

$$\text{NCC}_1 \geq (\text{MIN_NCC}_1 > 0) + \left\lfloor \frac{\alpha}{\beta} \right\rfloor + ((\alpha \bmod \beta) + 1 > \text{MAX_NCC}_2), \text{ with :}$$

$$\begin{cases} \bullet \alpha = \max(0, \text{NVERTEX_INITIAL} - \text{MIN_NCC}_1 - \text{MIN_NCC}_2 + 1), \\ \bullet \beta = \text{MAX_NCC}_1 + \text{MAX_NCC}_2 - 2. \end{cases}$$

(3.199)

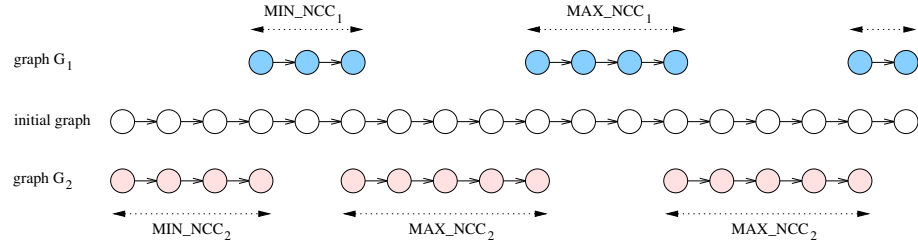


Figure 3.10: Illustration of Proposition 136. Configuration achieving the minimum number of connected components for G_1 according to the size of the smallest and largest connected components of G_1 and G_2 and to an initial number of vertices ($\text{MAX_NCC}_1 = 4$, $\text{MAX_NCC}_2 = 5$, $\text{MIN_NCC}_1 = 3$, $\text{MIN_NCC}_2 = 4$, $\text{NVERTEX_INITIAL} = 18$, $\alpha = \max(0, 18 - 3 - 4 + 1) = 12$, $\beta = \max(2, 4 + 5 - 2) = 7$, $\text{NCC}_1 = (3 > 0) + \lfloor \frac{12}{7} \rfloor + (((12 \bmod 7) + 1) > 5) = 3$)

Proof. The minimum number of connected components of G_1 is achieved by:

- Building a first connected component of G_2 involving MIN_NCC_2 vertices,
- Building a first connected component of G_1 involving MIN_NCC_1 vertices,

- Building alternatively a connected component of G_2 and a connected component of G_1 involving respectively MAX_NCC_2 and MAX_NCC_1 vertices,
- Finally, if this is possible, building a connected component of G_2 involving MAX_NCC_2 vertices and a connected component of G_1 with the remaining vertices. Note that these remaining vertices cannot be incorporated in the connected components previously built.

□

$$\boxed{\text{MAX_NCC}_1, \text{MAX_NCC}_2, \text{MIN_NCC}_1, \text{MIN_NCC}_2, \text{NCC}_2}$$

Proposition 137.

$$\begin{aligned} & \text{vpartition} \wedge \text{consecutive_loops_are_connected} : \\ & \text{MIN_NCC}_2 \cdot \max(0, \text{NCC}_2 - 1) + \text{MAX_NCC}_2 + \\ & \text{MIN_NCC}_1 \cdot \max(0, \text{NCC}_2 - 2) + \text{MAX_NCC}_1 \leq \text{NVERTEX}_{\text{INITIAL}} \end{aligned} \quad (3.200)$$

Proof. Similar to Proposition 130.

□

Proposition 138.

$$\begin{aligned} & \text{vpartition} \wedge \text{consecutive_loops_are_connected} : \\ & \overline{\text{NCC}_2} \leq (\overline{\text{MAX_NCC}_2} > 0) + \left\lfloor \frac{\alpha}{\beta} \right\rfloor + (\alpha \bmod \beta \geq \max(1, \overline{\text{MIN_NCC}_2})) \\ & \left\{ \begin{array}{l} \bullet \alpha = \max(0, \text{NVERTEX}_{\text{INITIAL}} - \max(1, \overline{\text{MAX_NCC}_2}) - \max(1, \overline{\text{MAX_NCC}_1})), \\ \bullet \beta = \max(1, \overline{\text{MIN_NCC}_2}) + \max(1, \overline{\text{MIN_NCC}_1}). \end{array} \right. \end{aligned} \quad (3.201)$$

Proof. Similar to Proposition 131.

□

Proposition 139.

$$\begin{aligned} & \text{vpartition} \wedge \text{consecutive_loops_are_connected} : \\ & \text{MAX_NCC}_2 \cdot \max(0, \text{NCC}_2 - 1) + \text{MIN_NCC}_2 + \\ & \text{MAX_NCC}_1 \cdot \text{NCC}_2 + \text{MIN_NCC}_1 \geq \text{NVERTEX}_{\text{INITIAL}} \end{aligned} \quad (3.202)$$

Proof. Similar to Proposition 132.

□

Proposition 140.

$$\begin{aligned} & \text{vpartition} \wedge \text{consecutive_loops_are_connected} : \\ & \text{NCC}_2 \geq (\overline{\text{MAX_NCC}_1} < \text{NVERTEX}_{\text{INITIAL}}) + \left\lfloor \frac{\alpha}{\beta} \right\rfloor + (\alpha \bmod \beta > \overline{\text{MAX_NCC}_1}) \\ & \left\{ \begin{array}{l} \bullet \alpha = \max(0, \text{NVERTEX}_{\text{INITIAL}} - \overline{\text{MIN_NCC}_2} - \overline{\text{MIN_NCC}_1}), \\ \bullet \beta = \max(1, \overline{\text{MAX_NCC}_2}) + \max(1, \overline{\text{MAX_NCC}_1}). \end{array} \right. \end{aligned} \quad (3.203)$$

Proof. Similar to Proposition 133.

□

Proposition 141.

$$\begin{aligned}
& \text{vpartition} \wedge \text{consecutive_loops_are_connected} : \\
& \text{MAX_NCC}_1 \leq \max(\text{MIN_NCC}_1, \text{NVERTEX}_{\text{INITIAL}} - \alpha), \text{ with :} \\
& \bullet \alpha = \text{MIN_NCC}_2 \cdot \max(0, \text{NCC}_2 - 1) + \text{MAX_NCC}_2 + \\
& \quad \text{MIN_NCC}_1 + \text{MIN_NCC}_1 \cdot \max(0, \text{NCC}_2 - 3)
\end{aligned} \tag{3.204}$$

Proof. Similar to Proposition 134. \square

Proposition 142.

$$\begin{aligned}
& \text{apartition} \wedge \text{arc_gen} = \text{PATH} \wedge \text{MIN_NCC}_1 > 1 \wedge \text{MIN_NCC}_2 > 1 : \\
& \text{NCC}_2 \leq (\text{MAX_NCC}_2 > 0) + \left\lfloor \frac{\alpha}{\beta} \right\rfloor + ((\alpha \bmod \beta) + 1 \geq \text{MIN_NCC}_2), \text{ with :} \\
& \left\{ \begin{array}{l} \bullet \alpha = \max(0, \text{NVERTEX}_{\text{INITIAL}} - \text{MAX_NCC}_1 - \text{MAX_NCC}_2 + 1), \\ \bullet \beta = \text{MIN_NCC}_1 + \text{MIN_NCC}_2 - 2. \end{array} \right.
\end{aligned} \tag{3.205}$$

Proof. Similar to Proposition 135. \square

Proposition 143.

$$\begin{aligned}
& \text{apartition} \wedge \text{arc_gen} = \text{PATH} \wedge \text{MIN_NCC}_1 > 1 \wedge \text{MIN_NCC}_2 > 1 : \\
& \text{NCC}_2 \geq (\text{MIN_NCC}_2 > 0) + \left\lfloor \frac{\alpha}{\beta} \right\rfloor + ((\alpha \bmod \beta) + 1 > \text{MAX_NCC}_1), \text{ with :} \\
& \left\{ \begin{array}{l} \bullet \alpha = \max(0, \text{NVERTEX}_{\text{INITIAL}} - \text{MIN_NCC}_1 - \text{MIN_NCC}_2 + 1), \\ \bullet \beta = \text{MAX_NCC}_1 + \text{MAX_NCC}_2 - 2. \end{array} \right.
\end{aligned} \tag{3.206}$$

Proof. Similar to Proposition 136. \square

Graph invariants relating six characteristics of two final graphs

$$\boxed{\text{MAX_NCC}_1, \text{MAX_NCC}_2, \text{MIN_NCC}_1, \text{MIN_NCC}_2, \text{NCC}_1, \text{NCC}_2}$$

Proposition 144.

$\text{apartition} \wedge \text{arc_gen} = \text{PATH} \wedge \text{NVERTEX}_{\text{INITIAL}} > 0 :$

$$\alpha \cdot \text{MIN_NCC}_1 + \text{MAX_NCC}_1 +$$

$$\beta \cdot \text{MIN_NCC}_2 + \text{MAX_NCC}_2 \leq \text{NVERTEX}_{\text{INITIAL}} + \text{NCC}_1 + \text{NCC}_2 - 1, \text{ with :}$$

$$\begin{cases} \bullet \alpha = \max(0, \text{NCC}_1 - 1), \\ \bullet \beta = \max(0, \text{NCC}_2 - 1). \end{cases}$$

(3.207)

Proof. Let $CC(G_1) = \{CC_a^1 : a \in [\text{NCC}_1]\}$ and $CC(G_2) = \{CC_a^2 : a \in [\text{NCC}_2]\}$ be respectively the set of connected components of the first and the second final graphs. Since the initial graph is a path, and since each arc of the initial graph belongs to the first or to the second final graphs (but not to both), there exists $(A_i)_{i \in [\text{NCC}_1 + \text{NCC}_2]}$ and there exists $j \in [2]$ such that $A_i \in CC(G_{1+(j \bmod 2)})$, for $i \bmod 2 = 0$ and $A_i \in CC(G_{1+((j+1) \bmod 2)})$ for $i \bmod 2 = 1$ and $A_i \cap A_{i+1} \neq \emptyset$ for $i \in [\text{NCC}_1 + \text{NCC}_2 - 1]$.

By inclusion-exclusion principle, since $A_i \cap A_j = \emptyset$ whenever $j \neq i + 1$, we obtain $\text{NVERTEX}_{\text{INITIAL}} = \sum_{a \in [\text{NCC}_1]} |CC_a^1| + \sum_{a \in [\text{NCC}_2]} |CC_a^2| - \sum_{i \in [\text{NCC}_1 + \text{NCC}_2 - 1]} |A_i \cap A_{i+1}|$. Since $|A_i \cap A_{i+1}|$ is equal to 1 for every well defined i , we obtain $\sum_{a \in [\text{NCC}_1]} |CC_a^1| + \sum_{a \in [\text{NCC}_2]} |CC_a^2| = \text{NVERTEX}_{\text{INITIAL}} + \text{NCC}_1 + \text{NCC}_2 - 1$.

Since $\alpha \cdot \text{MIN_NCC}_1 + \text{MAX_NCC}_1 + \beta \cdot \text{MIN_NCC}_2 + \text{MAX_NCC}_2 \leq \sum_{a \in [\text{NCC}_1]} |CC_a^1| + \sum_{a \in [\text{NCC}_2]} |CC_a^2|$ the result follows. \square

Proposition 145.

$\text{apartition} \wedge \text{arc_gen} = \text{PATH} \wedge \text{NVERTEX}_{\text{INITIAL}} > 0 :$

$$\alpha \cdot \text{MAX_NCC}_1 + \text{MIN_NCC}_1 +$$

$$\beta \cdot \text{MAX_NCC}_2 + \text{MIN_NCC}_2 \geq \text{NVERTEX}_{\text{INITIAL}} + \text{NCC}_1 + \text{NCC}_2 - 1, \text{ with :}$$

$$\begin{cases} \bullet \alpha = \max(0, \text{NCC}_1 - 1), \\ \bullet \beta = \max(0, \text{NCC}_2 - 1). \end{cases}$$

(3.208)

Proof. Similar to Proposition 144. \square

3.3 The electronic version of the catalog

An electronic version of the catalog containing every global constraint of the catalog is given in Appendix B. This electronic version was used for generating the \LaTeX file of this catalog, the figures associated with the graph-based description and a filtering algorithm for some of the constraints that use the automaton-based description. Within the electronic version, each constraint is described in terms of meta-data. A typical entry is:

```
ctr_date(minimum, ['20000128', '20030820', '20040530', '20041230']).

ctr_origin(minimum, 'CHIP', []).

ctr_arguments(minimum,
               ['MIN'-dvar
                'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(minimum,
                  [size('VARIABLES') > 0
                   required('VARIABLES', var)]).

ctr_graph(minimum,
           ['VARIABLES'],
           2,
           ['CLIQUE'>>collection(variables1, variables2)],
           [variables1^key = variables2^key #\ / variables1^var < variables2^var],
           ['ORDER' (0, 'MAXINT', var) = 'MIN']).

ctr_example(minimum,
             minimum(2, [[var-3], [var-2], [var-7], [var-2], [var-6]])).

ctr_see_also(minimum, [maximum]).

ctr_key_words(minimum, ['order constraint'
                        'minimum'
                        'maxint'
                        'automaton'
                        'automaton without counters'
                        'centered cyclic(1) constraint network(1)']).

ctr_automaton(minimum, minimum).

minimum(MIN, VARIABLES) :-
    minimum_signature(VARIABLES, SIGNATURE, MIN),
    automaton(SIGNATURE, _,
              SIGNATURE, 0..2,
              [source(s), node(e), sink(t)],
              [arc(s, 0, s), arc(s, 1, e),
               arc(e, 1, e), arc(e, 0, e), arc(e, $, t)],
              [], [], []).

minimum_signature([], [], _).
minimum_signature([[var-VAR]|VARs], [S|Ss], MIN) :-
    S in 0..2,
    MIN #< VAR #<=> S #= 0,
    MIN #= VAR #<=> S #= 1,
    MIN #> VAR #<=> S #= 2,
    minimum_signature(VARs, Ss, MIN).
```

and consists of the following Prolog facts, where `CONSTRAINT_NAME` is the name of the constraint under consideration. The facts are organized in the following 13 items:

- Items 1, 2, 5, 10 and 11 provide general information about a global constraint,

- Items 3, 4 and 6 describe the parameters of a global constraint.
- Items 7 and 8 describes the meaning of a global constraint in terms of a graph-based representation.
- Item 9 provides a ground instance which holds.
- Items 12 and 13 describe the meaning of a global constraint in term of an automaton-based representation.

Items 1, 2, 4 and 9 are mandatory, while all other items are optional. We now give the different items:

1. `ctr_date(CONSTRAINT_NAME, LIST_OF_DATES_OF_MODIFICATIONS)`

- `LIST_OF_DATES_OF_MODIFICATIONS` is a list of dates when the description of the constraint was modified.

2. `ctr_origin(CONSTRAINT_NAME, STRING, LIST_OF_CONSTRAINTS_NAMES)`

- `STRING` is a string denoting the origin of the constraint. `LIST_OF_CONSTRAINTS_NAMES` is an eventually empty list of constraint names related to the origin of the constraint.

3. `ctr_types(CONSTRAINT_NAME, LIST_OF_TYPES_DECLARATIONS)`

- `LIST_OF_TYPES_DECLARATIONS` is a list of elements of the form `name-type`, where `name` is the name of a new type and `type` the type itself (usually a collection). Basic and compound data types were respectively introduced in sections 1.1.1 and 1.1.2 page 3. This field is only used when we need to declare a new type that will be used for specifying the type of the arguments of the constraint. This is for instance the case when one argument of the constraint is a collection for which the type of one attribute is also a collection. This is for instance the case of the `diffn` constraint where the unique argument `ORTHOTOPE` is a collection of `ORTHOTOPE`; `ORTHOTOPE` refers to a new type declared in `LIST_OF_TYPES_DECLARATIONS`.

4. `ctr_arguments(CONSTRAINT_NAME, LIST_OF_ARGUMENTS_DECLARATIONS)`

- `LIST_OF_ARGUMENTS_DECLARATIONS` is a list of elements of the form `arg-type`, where `arg` is the name of an argument of the constraint and `type` the type of the argument. Basic and compound data types were respectively introduced in sections 1.1.1 and 1.1.2 page 3.

5. `ctr_synonyms(CONSTRAINT_NAME, LIST_OF_SYNONYMS)`

- `LIST_OF_SYNONYMS` is a list of synonyms for the constraint. This stems from the fact that, quite often, different authors use a different name for the same constraint. This is for instance the case for the `alldifferent` and the `symmetric_alldifferent` constraints.

6. `ctr_restrictions(CONSTRAINT_NAME, LIST_OF_RESTRICTIONS)`

- `LIST_OF_RESTRICTIONS` is a list of restrictions on the different argument of the constraint. Possible restrictions were described in Section 1.1.3 page 5.
7. `ctr_derived_collections(CONSTRAINT_NAME, LIST_OF_DERIVED_COLLECTIONS)`
- `LIST_OF_DERIVED_COLLECTIONS` is a list of derived collections. Derived collections are collections that are computed from the arguments of the constraint and are used in the graph-based description. Derived collections were described in Section 1.2.2 page 17.
8. `ctr_graph(CONSTRAINT_NAME, LIST_OF_ARC_INPUT, ARC_ARITY, ARC_GENERATORS, ARC_CONSTRAINTS, GRAPH_PROPERTIES)`
- `LIST_OF_ARC_INPUT` is a list of collections used for creating the vertices of the initial graph. This was described at page 43 of Section 1.2.3.
 - `ARC_ARITY` is the number of vertices of an arc. Arc arity was explained at page 44 of Section 1.2.3.
 - `ARC_GENERATORS` is a list of arc generators. Arc generators were introduced at page 43 of Section 1.2.3.
 - `ARC_CONSTRAINTS` is a list of arc constraints. Arc constraints were defined in Section 1.2.2 page 22.
 - `GRAPH_PROPERTIES` is a list of graph properties. Graph properties were described in Section 1.2.2 page 31.
9. `ctr_example(CONSTRAINT_NAME, LIST_OF_EXAMPLES)`
- `LIST_OF_EXAMPLES` is a list of examples (usually one). Each example corresponds to a ground instance for which the constraint holds.
10. `ctr_see_also(CONSTRAINT_NAME, LIST_OF_CONSTRAINTS)`
- `LIST_OF_CONSTRAINTS` is a list of constraints that are related in some way to the constraint.
11. `ctr_key_words(CONSTRAINT_NAME, LIST_OF_KEYWORDS)`
- `LIST_OF_KEYWORDS` is a list of keywords associated with the constraint. Keywords may be linked to the *meaning* of the constraint, to a *typical pattern* where the constraint can be applied or to a *specific problem* where the constraint is useful. All keywords used in the catalog are listed in alphabetic order in Section 2.5 page 62. Each keyword has an entry explaining its meaning and providing the list of global constraints using that keyword.
12. `ctr_automaton(CONSTRAINT_NAME, PREDICATE_NAME)`
- `PREDICATE_NAME` is the name of the Prolog predicate that creates the automata (usually one) associated with the constraint. This predicate name is usually the same as the constraint name, except for those constraints corresponding to a SICStus built-in (e.g. `in`, `element`).

13. `constraint_name(LIST_OF_ARGUMENTS) :- BODY:`

- `LIST_OF_ARGUMENTS` is the list of argument of the constraint.
- `BODY` corresponds to the Prolog code that creates the signature constraints as well as the automata (usually one) associated with the constraint. Within `BODY`, a fact of the form `automaton/9` describes the states and the transitions of the automata used for describing the set of solutions accepted by the constraint. It follows the description provided in Section 1.3.2 page 55.

Chapter 4

Global constraint catalog

Contents

4.1	all_differ_from_at_least_k_pos	172
4.2	all_min_dist	174
4.3	alldifferent	176
4.4	alldifferent_between_sets	180
4.5	alldifferent_except_0	182
4.6	alldifferent_interval	186
4.7	alldifferent_modulo	190
4.8	alldifferent_on_intersection	194
4.9	alldifferent_partition	198
4.10	alldifferent_same_value	200
4.11	allperm	204
4.12	among	206
4.13	among_diff_0	208
4.14	among_interval	212
4.15	among_low_up	214
4.16	among_modulo	218
4.17	among_seq	222
4.18	arith	224
4.19	arith_or	228
4.20	arith_sliding	232
4.21	assign_and_counts	234
4.22	assign_and_nvalues	238
4.23	atleast	242
4.24	atmost	246
4.25	balance	250
4.26	balance_interval	252
4.27	balance_modulo	256

4.28	balance_partition	260
4.29	bin_packing	264
4.30	binary_tree	268
4.31	cardinality_atleast	272
4.32	cardinality_atmost	276
4.33	cardinality_atmost_partition	280
4.34	change	284
4.35	change_continuity	288
4.36	change_pair	298
4.37	change_partition	302
4.38	circuit	306
4.39	circuit_cluster	310
4.40	circular_change	314
4.41	clique	318
4.42	colored_matrix	322
4.43	coloured_cumulative	324
4.44	coloured_cumulatives	328
4.45	common	332
4.46	common_interval	336
4.47	common_modulo	338
4.48	common_partition	340
4.49	connect_points	342
4.50	correspondence	346
4.51	count	350
4.52	counts	354
4.53	crossing	358
4.54	cumulative	362
4.55	cumulative_product	366
4.56	cumulative_two_d	370
4.57	cumulative_with_level_of_priority	374
4.58	cumulatives	378
4.59	cutset	382
4.60	cycle	386
4.61	cycle_card_on_path	390
4.62	cycle_or_accessibility	394
4.63	cycle_resource	398
4.64	cyclic_change	402
4.65	cyclic_change_joker	406
4.66	decreasing	410
4.67	deepest_valley	414
4.68	derangement	418
4.69	differ_from_at_least_k_pos	422
4.70	diffn	426

4.71	diffn_column	430
4.72	diffn_include	432
4.73	discrepancy	434
4.74	disjoint	436
4.75	disjoint_tasks	440
4.76	disjunctive	444
4.77	distance_between	446
4.78	distance_change	448
4.79	domain_constraint	452
4.80	elem	456
4.81	element	460
4.82	element_greatereq	464
4.83	element_lesseq	468
4.84	element_matrix	472
4.85	element_sparse	476
4.86	elements	480
4.87	elements_alldifferent	482
4.88	elements_sparse	486
4.89	eq_set	490
4.90	exactly	492
4.91	global_cardinality	496
4.92	global_cardinality_low_up	500
4.93	global_cardinality_with_costs	502
4.94	global_contiguity	506
4.95	golomb	508
4.96	graph_crossing	512
4.97	group	516
4.98	group_skip_isolated_item	524
4.99	highest_peak	532
4.100	in	536
4.101	in_relation	538
4.102	in_same_partition	542
4.103	in_set	546
4.104	increasing	548
4.105	indexed_sum	552
4.106	inflexion	554
4.107	int_value_precede	556
4.108	int_value_precede_chain	558
4.109	interval_and_count	560
4.110	interval_and_sum	564
4.111	inverse	568
4.112	inverse_set	572
4.113	ith_pos_different_from_0	576

4.114k_cut	578
4.115lex2	580
4.116lex_alldifferent	582
4.117lex_between	584
4.118lex_chain_less	588
4.119lex_chain_lesseq	592
4.120lex_different	596
4.121lex_greater	598
4.122lex_greatereq	602
4.123lex_less	606
4.124lex_lesseq	610
4.125link_set_to_booleans	614
4.126longest_change	618
4.127map	622
4.128max_index	624
4.129max_n	626
4.130max_nvalue	628
4.131max_size_set_of_consecutive_var	632
4.132maximum	634
4.133maximum_modulo	638
4.134min_index	640
4.135min_n	644
4.136min_nvalue	646
4.137min_size_set_of_consecutive_var	650
4.138minimum	652
4.139minimum_except_0	656
4.140minimum_greater_than	660
4.141minimum_modulo	664
4.142minimum_weight_alldifferent	666
4.143nclass	670
4.144nequivalence	674
4.145next_element	676
4.146next_greater_element	680
4.147ninterval	682
4.148no_peak	684
4.149no_valley	686
4.150not_all_equal	688
4.151not_in	690
4.152npair	694
4.153nset_of_consecutive_values	696
4.154nvalue	698
4.155nvalue_on_intersection	702
4.156nvalues	704

4.157	nvalues_except_0	706
4.158	one_tree	708
4.159	orchard	712
4.160	orth_link_ori_siz_end	716
4.161	orth_on_the_ground	718
4.162	orth_on_top_of_orth	720
4.163	orths_are_connected	722
4.164	path_from_to	726
4.165	pattern	730
4.166	peak	732
4.167	period	736
4.168	period_except_0	738
4.169	place_in_pyramid	740
4.170	polyomino	744
4.171	product_ctr	748
4.172	range_ctr	750
4.173	relaxed_sliding_sum	752
4.174	same	754
4.175	same_and_global_cardinality	760
4.176	same_intersection	764
4.177	same_interval	766
4.178	same_modulo	768
4.179	same_partition	770
4.180	sequence_folding	772
4.181	set_value_precede	776
4.182	shift	778
4.183	size_maximal_sequence_alldifferent	782
4.184	size_maximal_starting_sequence_alldifferent	784
4.185	sliding_card_skip0	786
4.186	sliding_distribution	790
4.187	sliding_sum	792
4.188	sliding_time_window	794
4.189	sliding_time_window_from_start	798
4.190	sliding_time_window_sum	802
4.191	smooth	806
4.192	soft_alldifferent_ctr	810
4.193	soft_alldifferent_var	814
4.194	soft_same_interval_var	818
4.195	soft_same_modulo_var	820
4.196	soft_same_partition_var	822
4.197	soft_same_var	824
4.198	soft_used_by_interval_var	828
4.199	soft_used_by_modulo_var	832

4.200	soft_used_by_partition_var	836
4.201	soft_used_by_var	838
4.202	sort	842
4.203	sort_permutation	846
4.204	stage_element	850
4.205	stretch_circuit	854
4.206	stretch_path	858
4.207	strict_lex2	862
4.208	strictly_decreasing	864
4.209	strictly_increasing	866
4.210	strongly_connected	868
4.211	sum	870
4.212	sum_ctr	874
4.213	sum_of_weights_of_distinct_values	876
4.214	sum_set	880
4.215	symmetric_alldifferent	882
4.216	symmetric_cardinality	886
4.217	symmetric_gcc	890
4.218	temporal_path	892
4.219	tour	896
4.220	track	900
4.221	tree	902
4.222	tree_range	906
4.223	tree_resource	910
4.224	two_layer_edge_crossing	914
4.225	two_orth_are_in_contact	918
4.226	two_orth_column	922
4.227	two_orth_do_not_overlap	924
4.228	two_orth_include	928
4.229	used_by	930
4.230	used_by_interval	934
4.231	used_by_modulo	936
4.232	used_by_partition	938
4.233	valley	940
4.234	vec_eq_tuple	944
4.235	weighted_partial_alldiff	946

4.1 all_differ_from_at_least_k_pos

Origin	Inspired by [56].
Constraint	<code>all_differ_from_at_least_k_pos(K, VECTORS)</code>
Type(s)	<code>VECTOR : collection(var - dvar)</code>
Argument(s)	<code>K : int</code> <code>VECTORS : collection(vec - VECTOR)</code>
Restriction(s)	<code>required(VECTOR, var)</code> <code>K ≥ 0</code> <code>required(VECTORS, vec)</code> <code>same_size(VECTORS, vec)</code>
Purpose	Enforce all pairs of distinct vectors of the <code>VECTORS</code> collection to differ from at least <code>K</code> positions.
Arc input(s)	<code>VECTORS</code>
Arc generator	$\text{CLIQUE}(\neq) \mapsto \text{collection}(\text{vectors1}, \text{vectors2})$
Arc arity	2
Arc constraint(s)	<code>differ_from_at_least_k_pos(K, vectors1.vec, vectors2.vec)</code>
Graph property(ies)	$\text{NARC} = \text{VECTORS} * \text{VECTORS} - \text{VECTORS} $

Example	$\text{all_differ_from_at_least_k_pos} \left(2, \left\{ \begin{array}{l} \text{vec} - \left\{ \begin{array}{l} \text{var} - 2, \\ \text{var} - 5, \\ \text{var} - 2, \\ \text{var} - 0 \end{array} \right\}, \\ \text{vec} - \left\{ \begin{array}{l} \text{var} - 3, \\ \text{var} - 6, \\ \text{var} - 2, \\ \text{var} - 1 \end{array} \right\}, \\ \text{vec} - \left\{ \begin{array}{l} \text{var} - 3, \\ \text{var} - 6, \\ \text{var} - 1, \\ \text{var} - 0 \end{array} \right\} \end{array} \right\} \right)$
----------------	--

The previous constraint holds since exactly $3 \cdot (3 - 1) = 6$ arc constraints hold, namely¹:

- The first and second vectors differ from 3 positions which is greater than or equal to $K = 2$.

¹Each item corresponds to two arc constraints.

- The first and third vectors differ from 3 positions which is greater than or equal to $K = 2$.
- The second and third vectors differ from 2 positions which is greater than or equal to $K = 2$.

Parts (A) and (B) of Figure 4.1 respectively show the initial and final graph. Since we use the **NARC** graph property, the arcs of the final graph are stressed in bold.

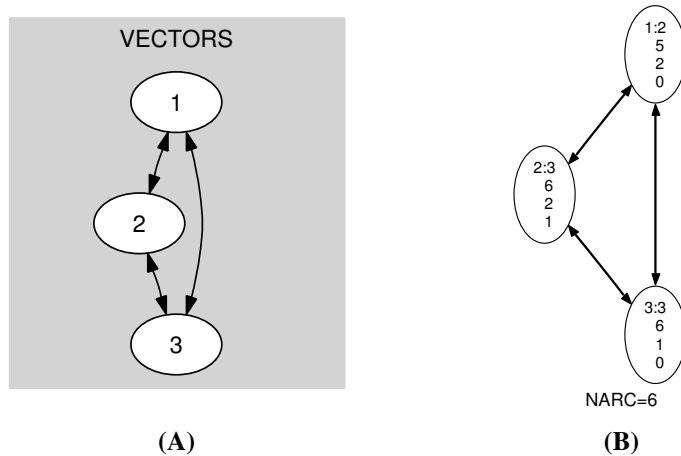


Figure 4.1: Initial and final graph of the `all_differ_from_at_least_k_pos` constraint

Graph model	The arc constraint(s) field uses the <code>differ_from_at_least_k_pos</code> constraint defined in this catalog.
Signature	Since we use the <code>CLIQUE(≠)</code> arc generator on the items of the <code>VECTORS</code> collection, the expression $ \text{VECTORS} \cdot \text{VECTORS} - \text{VECTORS} $ corresponds to the maximum number of arcs of the final graph. Therefore we can rewrite the graph property $\text{NARC} = \text{VECTORS} \cdot \text{VECTORS} - \text{VECTORS} $ to $\text{NARC} \geq \text{VECTORS} \cdot \text{VECTORS} - \text{VECTORS} $. This leads to simplify <u>NARC</u> to NARC .
See also	<code>differ_from_at_least_k_pos</code> .
Key words	decomposition, disequality, bioinformatics, vector, no_loop.

4.2 all_min_dist

Origin	[57]
Constraint	<code>all_min_dist(MINDIST, VARIABLES)</code>
Synonym(s)	<code>minimum_distance.</code>
Argument(s)	<code>MINDIST : int</code> <code>VARIABLES : collection(var - dvar)</code>
Restriction(s)	<code>MINDIST > 0</code> <code>required(VARIABLES, var)</code> <code>VARIABLES.var ≥ 0</code>
Purpose	Enforce for each pair $(\text{var}_i, \text{var}_j)$ of distinct variables of the collection <code>VARIABLES</code> that $ \text{var}_i - \text{var}_j \geq \text{MINDIST}$.
Arc input(s)	<code>VARIABLES</code>
Arc generator	$\text{CLIQUE}(<) \mapsto \text{collection}(\text{variables1}, \text{variables2})$
Arc arity	2
Arc constraint(s)	$\text{abs}(\text{variables1.var} - \text{variables2.var}) \geq \text{MINDIST}$
Graph property(ies)	$\text{NARC} = \text{VARIABLES} * (\text{VARIABLES} - 1) / 2$
Example	<code>all_min_dist (2, {var - 5, var - 1, var - 9, var - 3})</code>
	<p>Parts (A) and (B) of Figure 4.2 respectively show the initial and final graph. The <code>all_min_dist</code> constraint holds since all the arcs of the initial graph belong to the final graph: all the minimum distance constraints are satisfied.</p>
Graph model	<p>We generate a <i>clique</i> with a minimum distance constraint between each pair of distinct vertices and state that the number of arcs of the final graph should be equal to the number of arcs of the initial graph.</p>
Usage	<p>The <code>all_min_dist</code> constraint was initially created for handling frequency allocation problems.</p>
Remark	<p>The <code>all_min_dist</code> constraint can be modeled as a set of tasks which should not overlap. For each variable <code>var</code> of the <code>VARIABLES</code> collection we create a task t where <code>var</code> and <code>MINDIST</code> respectively correspond to the origin and the duration of t.</p>
See also	<code>alldifferent, diffn.</code>
Key words	value constraint, decomposition, frequency allocation problem.

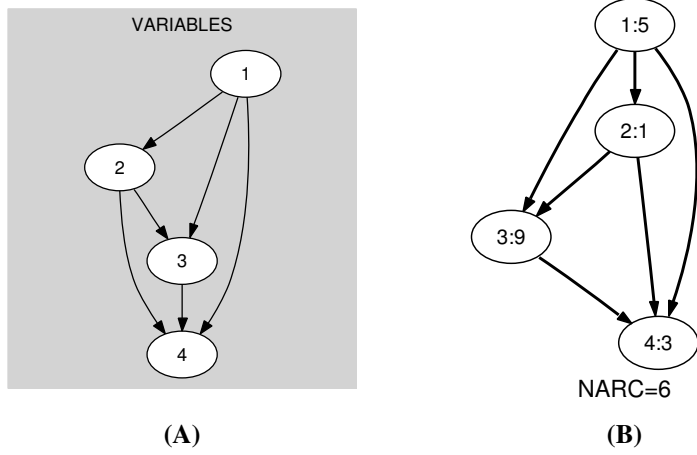


Figure 4.2: Initial and final graph of the `all_min_dist` constraint

4.3 alldifferent

Origin	[2]
Constraint	alldifferent(VARIABLES)
Synonym(s)	alldiff, alldistinct.
Argument(s)	VARIABLES : collection(var – dvar)
Restriction(s)	required(VARIABLES, var)
Purpose	Enforce all variables of the collection VARIABLES to take distinct values.
<hr/>	
Arc input(s)	VARIABLES
Arc generator	CLIQUE \mapsto collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	variables1.var = variables2.var
Graph property(ies)	MAX_NSCC \leq 1
<hr/>	
Example	alldifferent({var – 5, var – 1, var – 9, var – 3})
	Parts (A) and (B) of Figure 4.3 respectively show the initial and final graph. Since we use the MAX_NSCC graph property we show one of the largest strongly connected component of the final graph. The alldifferent holds since all the strongly connected components have at most one vertex: A value is used at most once.
Graph model	We generate a <i>clique</i> with an <i>equality</i> constraint between each pair of vertices (including a vertex and itself) and state that the size of the largest strongly connected component should not exceed one.
Automaton	Figure 4.4 depicts the automaton associated to the alldifferent constraint. To each item of the collection VARIABLES corresponds a signature variable S_i , which is equal to 1. The automaton counts the number of occurrences of each value and finally imposes that each value is taken at most one time.
Usage	The alldifferent constraint occurs in most practical problems. A classical example is the n -queen chess puzzle problem: Place n queens on a n by n chessboard in such a way that no queen attacks another. Two queens attack each other if they are located on the same column, on the same row or on the same diagonal. This can be modelled as the conjunction of three alldifferent constraints. We associate to the i^{th} column of the chessboard a domain variable X_i that gives the line number where the corresponding queen is located. The three alldifferent constraints are:

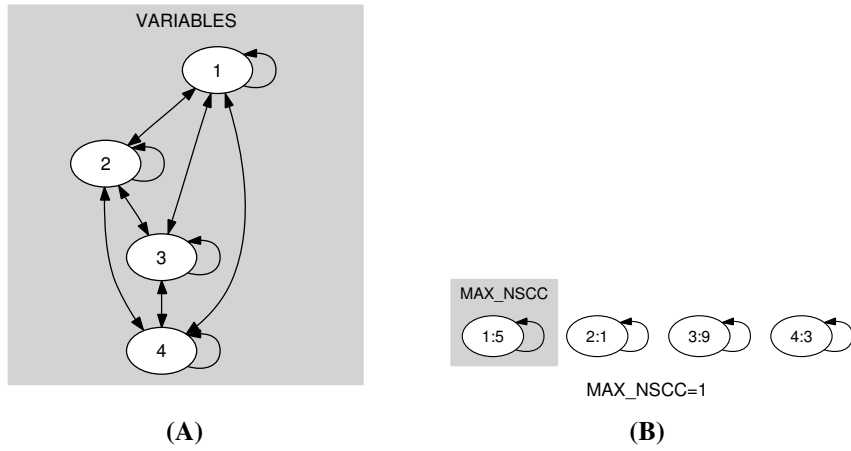


Figure 4.3: Initial and final graph of the alldifferent constraint

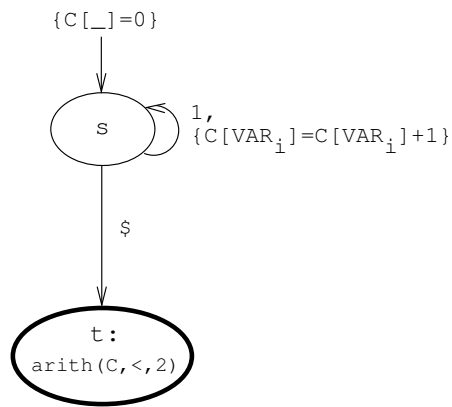


Figure 4.4: Automaton of the alldifferent constraint

- `alldifferent`($X_1, X_2 + 1, \dots, X_n + n - 1$) for the upper-left to lower-right diagonals,
- `alldifferent`(X_1, X_2, \dots, X_n) for the lines,
- `alldifferent`($X_1 + n - 1, X_2 + n - 2, \dots, X_n$) for the lower-right to upper-left diagonals.

They are respectively depicted by parts (A), (C) and (D) of Figure 4.5.

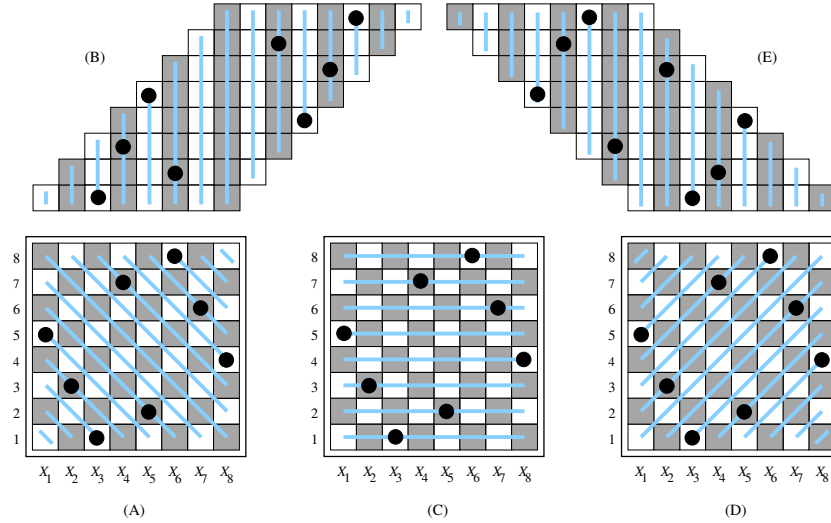


Figure 4.5: Upper-left to lower-right diagonals (A-B), lines (C) and lower-right to upper-left diagonals (D-E)

Remark

Even if the `alldifferent` constraint had not this form, it was specified in ALICE [58, 2] by asking for an injective correspondence between variables and values: $x \neq y \Rightarrow f(x) \neq f(y)$.

For possible relaxations of the `alldifferent` constraints see the `alldifferent_except_0`, the `soft_alldifferent_ctr`, the `soft_alldifferent_var` and the `weighted_partial_alldiff` constraints.

Algorithm

The first complete filtering algorithm was independently found by Marie-Christine Costa [59] and Jean-Charles Régin [18]. This algorithm is based on a corollary of Claude Berge which characterizes the edges of a graph that belong to a maximum matching but not to all [17, page 120]. A short time after, assuming that all variables have no holes in their domain, Michel Leconte came up with a filtering algorithm [60] based on edge finding. A first bound-consistency algorithm was proposed by Bleuzen-Guernalec et al. [61]. Later on, two different approaches were used to design bound-consistency algorithms. Both approaches model the constraint as a bipartite graph. The first identifies Hall intervals in this graph [62, 63] and the second applies the same algorithm that is used to compute arc-consistency, but achieves a speedup by exploiting the simpler structure of the graph [23].

Used in

`circuit_cluster`, `correspondence`, `size_maximal_sequence_alldifferent`, `size_maximal_starting_sequence_alldifferent`, `sort_permutation`.

See also alldifferent_except_0, soft_alldifferent_var, soft_alldifferent_ctr,
cycle, symmetric_alldifferent, lex_alldifferent,
alldifferent_on_intersection, weighted_partial_alldiff.

Key words value constraint, permutation, all different, disequality, bipartite matching, n-queen,
Hall interval, bound-consistency, automaton, automaton with array of counters, one_succ.

4.4 alldifferent_between_sets

Origin	ILOG
Constraint	alldifferent_between_sets(VARIABLES)
Synonym(s)	all_null_intersect, alldiff_between_sets, alldistinct_between_sets.
Argument(s)	VARIABLES : collection(var – svar)
Restriction(s)	required(VARIABLES, var)
Purpose	Enforce all sets of the collection VARIABLES to be distinct.
<hr/>	
Arc input(s)	VARIABLES
Arc generator	<i>CLIQUE</i> \mapsto collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	eq_set(variables1.var, variables2.var)
Graph property(ies)	$MAX_NSCC \leq 1$
<hr/>	
Example	$\text{alldifferent_between_sets} \left(\left\{ \begin{array}{l} \text{var} - \{3, 5\}, \\ \text{var} - \emptyset, \\ \text{var} - \{3\}, \\ \text{var} - \{3, 5, 7\} \end{array} \right\} \right)$ <p>Parts (A) and (B) of Figure 4.6 respectively show the initial and final graph. Since we use the MAX_NSCC graph property we show one of the largest strongly connected component of the final graph. The <code>alldifferent_between_sets</code> holds since all the strongly connected components have at most one vertex.</p>
Graph model	We generate a <i>clique</i> with binary <i>set equalities</i> constraints between each pair of vertices (including a vertex and itself) and state that the size of the largest strongly connected component should not exceed one.
Usage	This constraint is available in some configuration library offered by Ilog.
See also	alldifferent, link_set_to_booleans.
Key words	all different, disequality, bipartite matching, constraint involving set variables, one_succ.

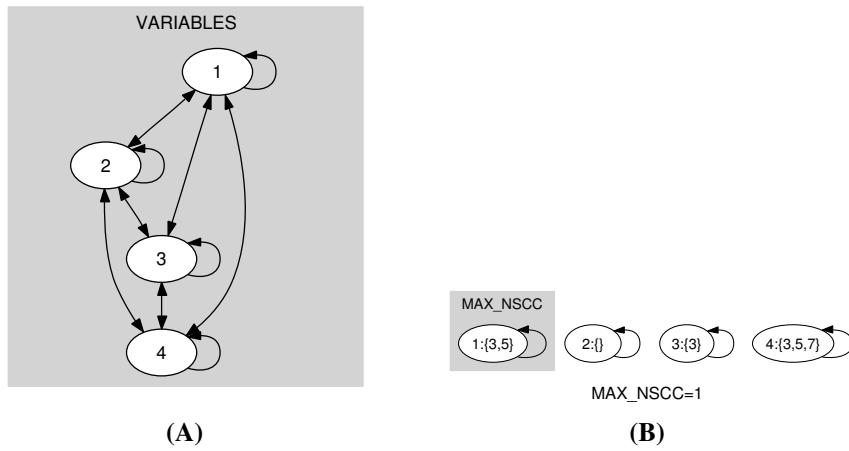


Figure 4.6: Initial and final graph of the `alldifferent_between_sets` constraint

4.5 alldifferent_except_0

Origin	Derived from alldifferent.
Constraint	alldifferent_except_0(VARIABLES)
Synonym(s)	alldiff_except_0, alldistinct_except_0.
Argument(s)	VARIABLES : collection(var – dvar)
Restriction(s)	required(VARIABLES, var)
Purpose	Enforce all variables of the collection VARIABLES to take distinct values, except those variables which are assigned to 0.
Arc input(s)	VARIABLES
Arc generator	<i>CLIQUE</i> \mapsto collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	<ul style="list-style-type: none"> • variables1.var \neq 0 • variables1.var = variables2.var
Graph property(ies)	$\text{MAX_NSCC} \leq 1$
Example	$\text{alldifferent_except_0} \left(\left(\begin{array}{l} \text{var} - 5, \\ \text{var} - 0, \\ \text{var} - 1, \\ \text{var} - 9, \\ \text{var} - 0, \\ \text{var} - 3 \end{array} \right) \right)$ <p>Parts (A) and (B) of Figure 4.7 respectively show the initial and final graph. Since we use the MAX_NSCC graph property we show one of the largest strongly connected component of the final graph. The alldifferent_except_0 holds since all the strongly connected components have at most one vertex: A value different from 0 is used at most once.</p>
Graph model	The graph model is the same as the one used for the alldifferent constraint, except that we discard all variables that are assigned to 0.
Automaton	Figure 4.8 depicts the automaton associated to the alldifferent_except_0 constraint. To each variable VAR_i of the collection VARIABLES corresponds a 0-1 signature variable S_i . The following signature constraint links VAR_i and S_i : $\text{VAR}_i \neq 0 \Leftrightarrow S_i$. The automaton counts the number of occurrences of each value different from 0 and finally imposes that each non-zero value is taken at most one time.

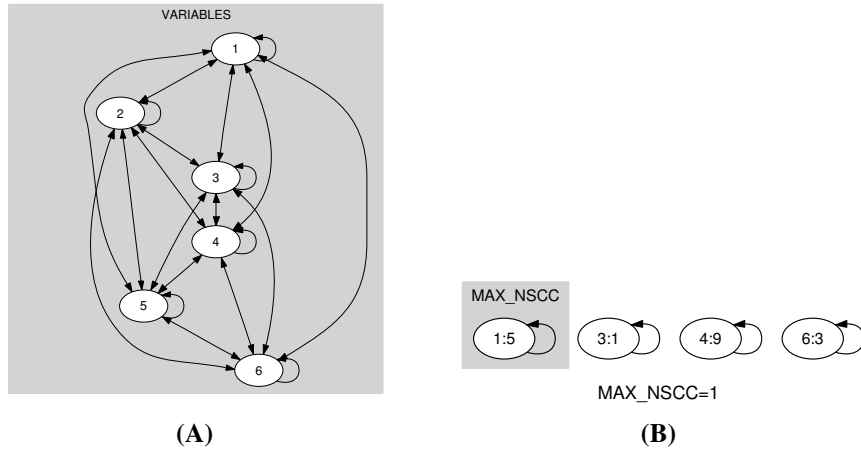


Figure 4.7: Initial and final graph of the alldifferent_except_0 constraint

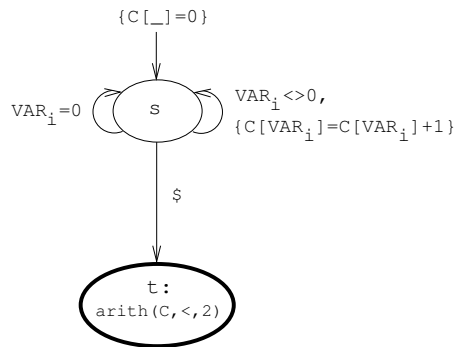


Figure 4.8: Automaton of the alldifferent_except_0 constraint

- Usage** Quite often it appears that for some modelling reason you create a *joker value*. You don't want that normal constraints hold for variables that take this *joker value*. For this purpose we modify the binary arc constraint in order to discard the vertices for which the corresponding variables are assigned to 0. This will be effectively the case since all the corresponding arcs constraints will not hold.
- See also** `alldifferent`, `weighted_partial_alldiff`.
- Key words** value constraint, relaxation, joker value, all different, automaton, automaton with array of counters, `one_succ`.

4.6 alldifferent_interval

Origin	Derived from <code>alldifferent</code> .
Constraint	<code>alldifferent_interval(VARIABLES, SIZE_INTERVAL)</code>
Synonym(s)	<code>alldiff_interval</code> , <code>alldistinct_interval</code> .
Argument(s)	<pre>VARIABLES : collection(var - dvar) SIZE_INTERVAL : int</pre>
Restriction(s)	<pre>required(VARIABLES, var) SIZE_INTERVAL > 0</pre>
Purpose	<p>Enforce all variables of the collection <code>VARIABLES</code> to belong to distinct intervals. The intervals are defined by $[SIZE_INTERVAL \cdot k, SIZE_INTERVAL \cdot k + SIZE_INTERVAL - 1]$ where k is an integer.</p>
Arc input(s)	<code>VARIABLES</code>
Arc generator	<code>CLIQUE</code> \mapsto <code>collection(variables1, variables2)</code>
Arc arity	2
Arc constraint(s)	<code>variables1.var/SIZE_INTERVAL = variables2.var/SIZE_INTERVAL</code>
Graph property(ies)	<code>MAX_NSCC \leq 1</code>
Example	<pre>alldifferent_interval({var - 2, var - 3, var - 10}, 3)</pre> <p>In the previous example, the second parameter <code>SIZE_INTERVAL</code> defines the following family of intervals $[3 \cdot k, 3 \cdot k + 2]$, where k is an integer. Since the three variables of the collection <code>VARIABLES</code> take values that are respectively located within the three following distinct intervals $[0, 2]$, $[3, 5]$ and $[9, 11]$, the <code>alldifferent_interval</code> constraint holds. Parts (A) and (B) of Figure 4.9 respectively show the initial and final graph. Since we use the <code>MAX_NSCC</code> graph property we show one of the largest strongly connected component of the final graph.</p>
Graph model	<p>Similar to the <code>alldifferent</code> constraint, but we replace the binary <i>equality</i> constraint of the <code>alldifferent</code> constraint by the fact that two variables are respectively assigned to two values that belong to the same interval. We generate a <i>clique</i> with a <i>belong to the same interval</i> constraint between each pair of vertices (including a vertex and itself) and state that the size of the largest strongly connected component should not exceed one.</p>
Automaton	<p>Figure 4.10 depicts the automaton associated to the <code>alldifferent_interval</code> constraint. To each item of the collection <code>VARIABLES</code> corresponds a signature variable S_i, which is equal to 1. For each interval $[SIZE_INTERVAL \cdot k, SIZE_INTERVAL \cdot k + SIZE_INTERVAL - 1]$ of values the automaton counts the number of occurrences of its values and finally imposes that the values of an interval are taken at most once.</p>

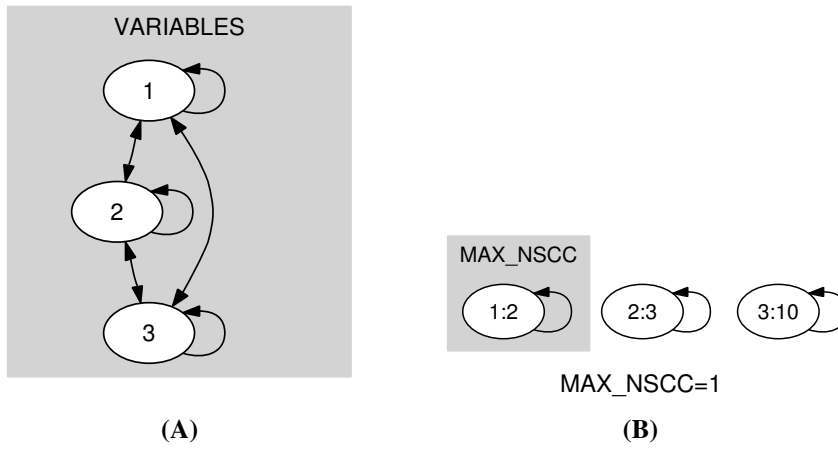


Figure 4.9: Initial and final graph of the `alldifferent_interval` constraint

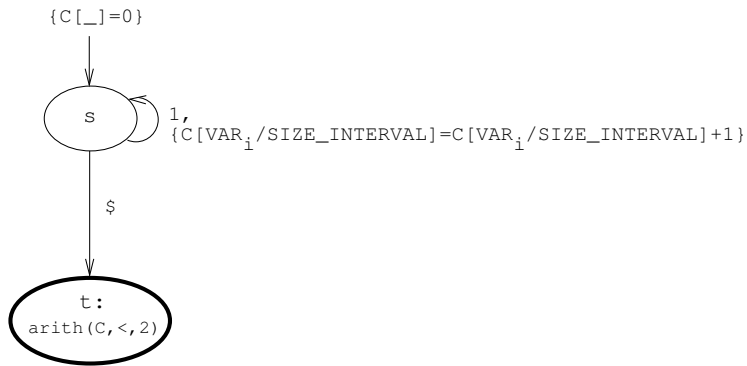


Figure 4.10: Automaton of the `alldifferent_interval` constraint

See also alldifferent.

Key words value constraint, interval, all different, automaton, automaton with array of counters, one_succ.

4.7 alldifferent_modulo

Origin	Derived from <code>alldifferent</code> .
Constraint	<code>alldifferent_modulo(VARIABLES, M)</code>
Synonym(s)	<code>alldiff_modulo</code> , <code>alldistinct_modulo</code> .
Argument(s)	<p>VARIABLES : collection(var – dvar)</p> <p>M : int</p>
Restriction(s)	<p>required(VARIABLES, var)</p> <p>$M \neq 0$</p> <p>$M \geq \text{VARIABLES}$</p>
Purpose	Enforce all variables of the collection <code>VARIABLES</code> to have a distinct rest when divided by <code>M</code> .
<hr/>	
Arc input(s)	VARIABLES
Arc generator	$CLIQUE \mapsto \text{collection}(\text{variables1}, \text{variables2})$
Arc arity	2
Arc constraint(s)	$\text{variables1.var} \bmod M = \text{variables2.var} \bmod M$
Graph property(ies)	$\text{MAX_NSCC} \leq 1$
<hr/>	
Example	$\text{alldifferent_modulo} \left(\left(\left\{ \begin{array}{l} \text{var} - 25, \\ \text{var} - 1, \\ \text{var} - 14, \\ \text{var} - 3 \end{array} \right\}, 5 \right) \right)$ <p>The equivalence classes associated to values 25, 1, 14 and 3 are respectively equal to $25 \bmod 5 = 0$, $1 \bmod 5 = 1$, $14 \bmod 5 = 4$ and $3 \bmod 5 = 3$. Since they are distinct the <code>alldifferent_modulo</code> constraint holds. Parts (A) and (B) of Figure 4.11 respectively show the initial and final graph. Since we use the <code>MAX_NSCC</code> graph property we show one of the largest strongly connected component of the final graph.</p>
Graph model	Exploit the same model used for the <code>alldifferent</code> constraint. We replace the binary <i>equality</i> constraint by an other equivalence relation depicted by the arc constraint. We generate a <i>clique</i> with a binary <i>equality modulo M</i> constraint between each pair of vertices (including a vertex and itself) and state that the size of the largest strongly connected component should not exceed one.
Automaton	Figure 4.12 depicts the automaton associated to the <code>alldifferent_modulo</code> constraint. To each item of the collection <code>VARIABLES</code> corresponds a signature variable S_i , which is equal to 1. The automaton counts for each equivalence class the number of used values and finally imposes that each equivalence class is used at most one time.

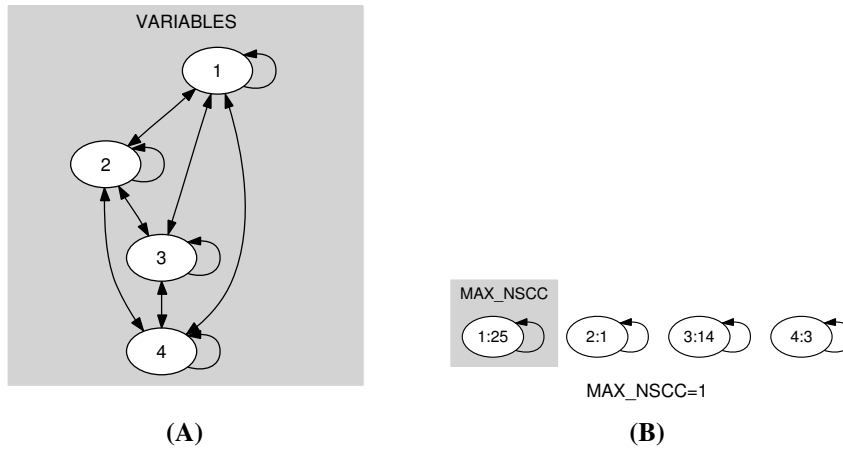


Figure 4.11: Initial and final graph of the alldifferent_modulo constraint

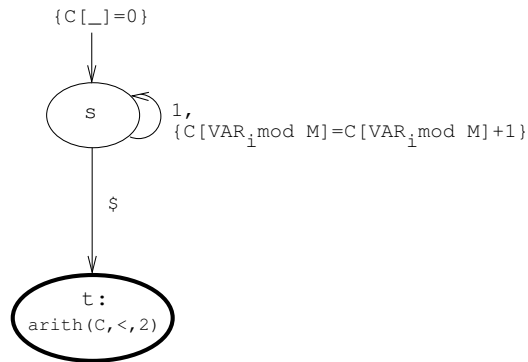


Figure 4.12: Automaton of the alldifferent_modulo constraint

See also alldifferent.

Key words value constraint, modulo, all different, automaton, automaton with array of counters, one_succ.

4.8 alldifferent_on_intersection

Origin	Derived from common and alldifferent.
Constraint	alldifferent_on_intersection(VARIABLES1, VARIABLES2)
Synonym(s)	alldiff_on_intersection, alldistinct_on_intersection.
Argument(s)	VARIABLES1 : collection(var - dvar) VARIABLES2 : collection(var - dvar)
Restriction(s)	required(VARIABLES1, var) required(VARIABLES2, var)
Purpose	The values which both occur in the VARIABLES1 and VARIABLES2 collections have only one occurrence.
Arc input(s)	VARIABLES1 VARIABLES2
Arc generator	<i>PRODUCT</i> \mapsto collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	variables1.var = variables2.var
Graph property(ies)	$MAX_NCC \leq 2$

Example	$\text{alldifferent_on_intersection} \left(\left(\left\{ \begin{array}{l} \text{var} - 5, \\ \text{var} - 9, \\ \text{var} - 1, \\ \text{var} - 5 \end{array} \right\}, \right. \right. \\ \left. \left. \left\{ \begin{array}{l} \text{var} - 2, \\ \text{var} - 1, \\ \text{var} - 6, \\ \text{var} - 9, \\ \text{var} - 6, \\ \text{var} - 2 \end{array} \right\} \right) \right)$
----------------	---

Parts (A) and (B) of Figure 4.13 respectively show the initial and final graph. Since we use the *MAX_NCC* graph property we show one of the largest connected component of the final graph. The *alldifferent_on_intersection* constraint holds since each connected component has at most two vertices. Observe that all the vertices corresponding to the variables that take values 5, 2 or 6 were removed from the final graph since there is no arc for which the associated equality constraint holds.

Automaton Figure 4.14 depicts the automaton associated to the *alldifferent_on_intersection* constraint. To each variable $VAR1_i$ of the collection *VARIABLES1* corresponds a signature variable S_i , which is equal to 0. To each variable $VAR2_i$ of the collection *VARIABLES2* corresponds a signature variable $S_{i+|VARIABLES1|}$, which is equal to 1. The automaton first counts

the number of occurrences of each value assigned to the variables of the VARIABLES1 collection. It then counts the number of occurrences of each value assigned to the variables of the VARIABLES2 collection. Finally, the automaton imposes that each value is not taken by two variables of both collections.

See also

`alldifferent`, `common`, `nvalue_on_intersection`, `same_intersection`.

Key words

value constraint, all different, connected component, constraint on the intersection, automaton, automaton with array of counters, acyclic, bipartite, no_loop.

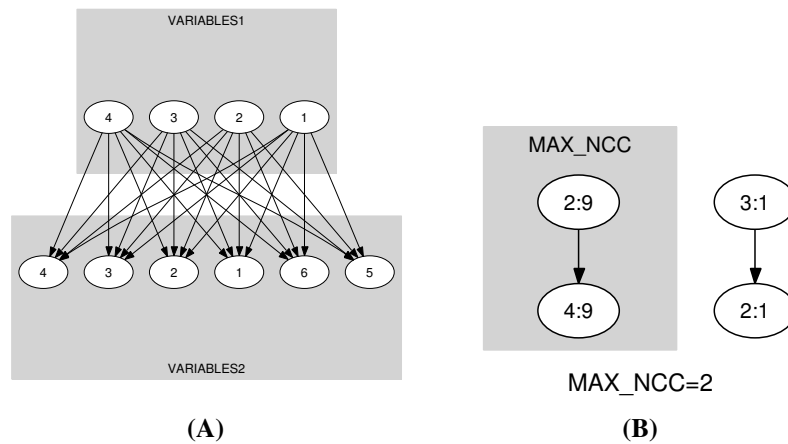


Figure 4.13: Initial and final graph of the `alldifferent_on_intersection` constraint

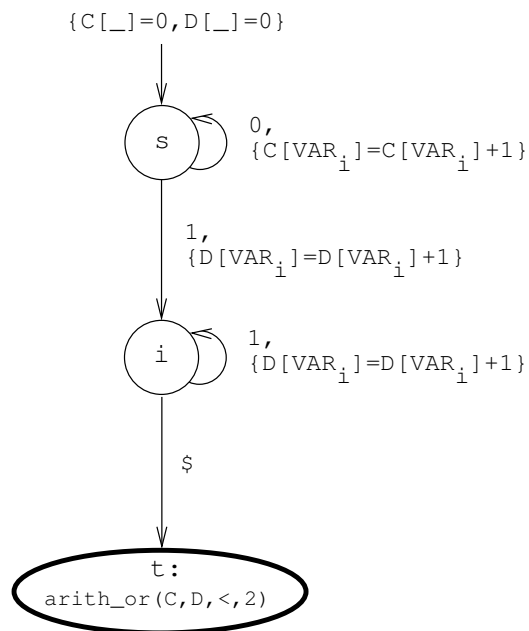


Figure 4.14: Automaton of the `alldifferent_on_intersection` constraint

4.9 alldifferent_partition

Origin	Derived from <code>alldifferent</code> .
Constraint	<code>alldifferent_partition(VARIABLES, PARTITIONS)</code>
Synonym(s)	<code>alldiff_partition</code> , <code>alldistinct_partition</code> .
Type(s)	VALUES : <code>collection(val - int)</code>
Argument(s)	VARIABLES : <code>collection(var - dvar)</code> PARTITIONS : <code>collection(p - VALUES)</code>
Restriction(s)	<code>required(VALUES, val)</code> <code>distinct(VALUES, val)</code> $ VARIABLES \leq PARTITIONS $ <code>required(VARIABLES, var)</code> $ PARTITIONS \geq 2$ <code>required(PARTITIONS, p)</code>
Purpose	Enforce all variables of the collection <code>VARIABLES</code> to take values which belong to distinct partitions.
Arc input(s)	VARIABLES
Arc generator	<code>CLIQUE</code> \mapsto <code>collection(variables1, variables2)</code>
Arc arity	2
Arc constraint(s)	<code>in_same_partition(variables1.var, variables2.var, PARTITIONS)</code>
Graph property(ies)	$MAX_NSCC \leq 1$
Example	$\text{alldifferent_partition} \left(\begin{array}{l} \{ \text{var} - 6, \text{var} - 3, \text{var} - 4 \}, \\ \left\{ \begin{array}{l} \text{p} - \{ \text{val} - 1, \text{val} - 3 \}, \\ \text{p} - \{ \text{val} - 4 \}, \\ \text{p} - \{ \text{val} - 2, \text{val} - 6 \} \end{array} \right\} \end{array} \right)$ <p>Since all variables take values that are located within distinct partitions the <code>alldifferent_partition</code> constraint holds. Parts (A) and (B) of Figure 4.15 respectively show the initial and final graph. Since we use the <code>MAX_NSCC</code> graph property we show one of the largest strongly connected component of the final graph.</p>
Graph model	Similar to the <code>alldifferent</code> constraint, but we replace the binary <i>equality</i> constraint of the <code>alldifferent</code> constraint by the fact that two variables are respectively assigned to two values that belong to the same partition. We generate a <i>clique</i> with a <code>in_same_partition</code> constraint between each pair of vertices (including a vertex and itself) and state that the size of the largest strongly connected component should not exceed one.
See also	<code>alldifferent</code> , <code>in_same_partition</code> .
Key words	value constraint, partition, all different, one_succ.

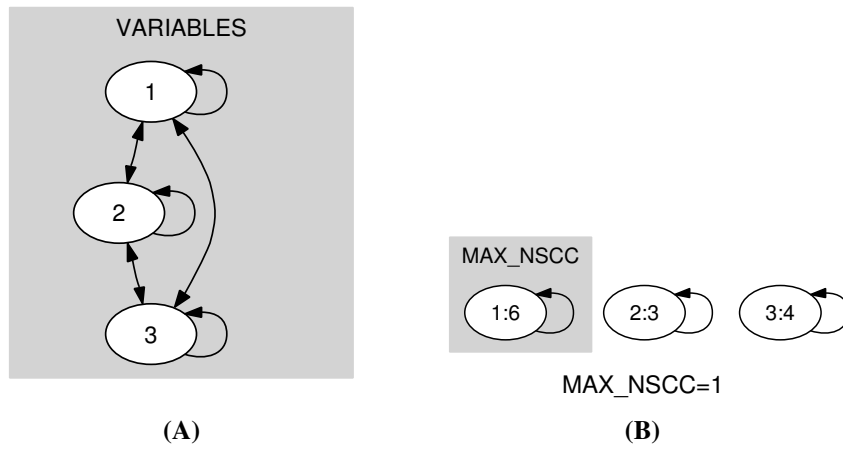


Figure 4.15: Initial and final graph of the `alldifferent_partition` constraint

4.10 alldifferent_same_value

Origin	Derived from alldifferent.
Constraint	alldifferent_same_value(NSAME, VARIABLES1, VARIABLES2)
Synonym(s)	alldiff_same_value, alldistinct_same_value.
Argument(s)	NSAME : dvar VARIABLES1 : collection(var – dvar) VARIABLES2 : collection(var – dvar)
Restriction(s)	NSAME ≥ 0 NSAME ≤ VARIABLES1 VARIABLES1 = VARIABLES2 required(VARIABLES1, var) required(VARIABLES2, var)
Purpose	All the values assigned to the variables of the collection VARIABLES1 are pairwise distinct. NSAME is equal to number of constraints of the form VARIABLES1[i].var = VARIABLES2[i].var (1 ≤ i ≤ VARIABLES1) that hold.
Arc input(s)	VARIABLES1 VARIABLES2
Arc generator	<i>PRODUCT(CLIQUE, LOOP, =)</i> ↦ collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	variables1.var = variables2.var
Graph property(ies)	<ul style="list-style-type: none"> • MAX_NSCC ≤ 1 • NARC_NO_LOOP = NSAME

Example	$\text{alldifferent_same_value} \left(2, \left(\begin{array}{c} \left\{ \begin{array}{c} \text{var} - 7, \\ \text{var} - 3, \\ \text{var} - 1, \\ \text{var} - 5 \end{array} \right\}, \\ \left\{ \begin{array}{c} \text{var} - 1, \\ \text{var} - 3, \\ \text{var} - 1, \\ \text{var} - 7 \end{array} \right\} \end{array} \right) \right)$
----------------	--

Part (A) of Figure 4.16 gives the initial graph that is generated. Variables of collection VARIABLES1 are coloured, while variables of collection VARIABLES2 are kept in white. Part (B) represents the final graph associated to the example. In this graph each vertex constitutes a strongly connected component and the number of arcs that do not correspond to a loop is equal to 2 (i.e. NSAME).

Graph model

The arc generator $PRODUCT(CLIQUE, LOOP, =)$ is used in order to generate all the arcs of the initial graph:

- The arc generator $CLIQUE$ creates all links between the items of the first collection $VARIABLES1$,
- The arc generator $LOOP$ creates one loop for all items of the second collection $VARIABLES2$,
- Finally the arc generator $PRODUCT(=)$ creates an arc between items located at the same position in the collections $VARIABLES1$ and $VARIABLES2$.

Automaton

Figure 4.17 depicts the automaton associated to the `alldifferent_same_value` constraint. Let $VAR1_i$ and $VAR2_i$ respectively denote the i^{th} variables of the $VARIABLES1$ and $VARIABLES2$ collections. To each pair of variables $(VAR1_i, VAR2_i)$ corresponds a signature variable S_i . The following signature constraint links $VAR1_i$, $VAR2_i$ and S_i : $VAR1_i = VAR2_i \Leftrightarrow S_i$.

Usage

When all variables of the second collection are initially bound to distinct values the `alldifferent_same_value` constraint can be explained in the following way:

- We interpret the variables of the second collection as the previous solution of a problem where all variables have to be distinct.
- We interpret the variables of the first collection as the current solution to find, where all variables should again be pairwise distinct.

The variable `NSAME` measures the `distance` of the current solution from the previous solution. This corresponds to the number of variables of $VARIABLES2$ that are not assigned to the same previous value.

Key words

proximity constraint, automaton, automaton with array of counters.

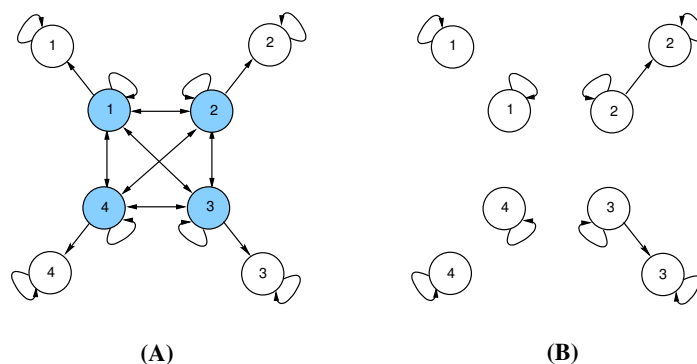
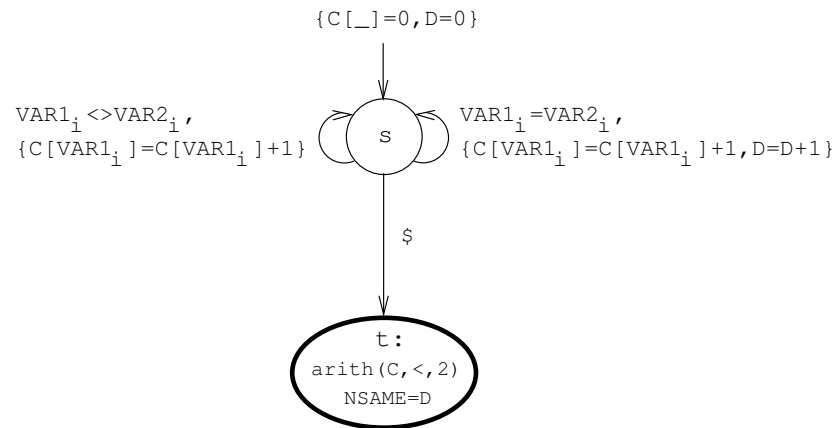


Figure 4.16: Initial and final graph of the `alldifferent_same_value` constraint

Figure 4.17: Automaton of the `alldifferent_same_value` constraint

20000128

203

4.11 allperm

Origin	[64]
Constraint	allperm(MATRIX)
Type(s)	VECTOR : collection(var – dvar)
Argument(s)	MATRIX : collection(vec – VECTOR)
Restriction(s)	required(VECTOR, var) required(MATRIX, vec) same_size(MATRIX, vec)
Purpose	Given a matrix of domain variables, enforces that the first row is lexicographically less than or equal to all permutations of all other rows.
Example	$\text{allperm} \left(\left\{ \begin{array}{l} \text{vec} - \{\text{var} - 1, \text{var} - 2, \text{var} - 3\}, \\ \text{vec} - \{\text{var} - 3, \text{var} - 1, \text{var} - 2\} \end{array} \right\} \right)$ <p>The previous constraint holds since vector $\langle 1, 2, 3 \rangle$ is lexicographically less than or equal to all the permutations of vector $\langle 3, 1, 2 \rangle$ (i.e. $\langle 1, 2, 3 \rangle$, $\langle 1, 3, 2 \rangle$, $\langle 2, 1, 3 \rangle$, $\langle 2, 3, 1 \rangle$, $\langle 3, 1, 2 \rangle$, $\langle 3, 2, 1 \rangle$).</p>
Usage	A <i>symmetry-breaking</i> constraint.
See also	lex2, lex_lesseq.
Key words	predefined constraint, order constraint, matrix, matrix model, symmetry, lexicographic order.

4.12 among

Origin	[37]
Constraint	<code>among(NVAR, VARIABLES, VALUES)</code>
Argument(s)	<code>NVAR</code> : dvar <code>VARIABLES</code> : <code>collection(var - dvar)</code> <code>VALUES</code> : <code>collection(val - int)</code>
Restriction(s)	<code>NVAR</code> \geq 0 <code>NVAR</code> \leq <code> VARIABLES </code> <code>required(VARIABLES, var)</code> <code>required(VALUES, val)</code> <code>distinct(VALUES, val)</code>
Purpose	<code>NVAR</code> is the number of variables of the collection <code>VARIABLES</code> which take their value in <code>VALUES</code> .
Arc input(s)	<code>VARIABLES</code>
Arc generator	<code>SELF</code> \mapsto <code>collection(variables)</code>
Arc arity	1
Arc constraint(s)	<code>in(variables.var, VALUES)</code>
Graph property(ies)	<code>NARC = NVAR</code>

Example

$$\text{among} \left(3, \left\{ \begin{array}{l} \text{var} - 4, \\ \text{var} - 5, \\ \text{var} - 5, \\ \text{var} - 4, \\ \text{var} - 1 \end{array} \right\}, \{ \text{val} - 1, \text{val} - 5, \text{val} - 8 \} \right)$$

Parts (A) and (B) of Figure 4.18 respectively show the initial and final graph. Since we use the **NARC** graph property, the unary arcs of the final graph are stressed in bold.

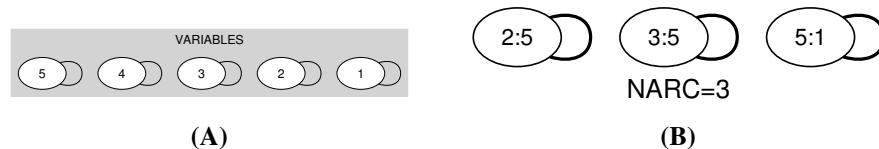


Figure 4.18: Initial and final graph of the among constraint

Graph model

The arc constraint corresponds to the unary constraint `in(variables.var, VALUES)` defined in this catalog. Since this is a unary constraint we employ the *SELF* arc generator in order to produce an initial graph with a single loop on each vertex.

Automaton

Figure 4.19 depicts the automaton associated to the `among` constraint. To each variable VAR_i of the collection `VARIABLES` corresponds a 0-1 signature variable S_i . The following signature constraint links VAR_i and S_i : $VAR_i \in \text{VALUES} \Leftrightarrow S_i$. The automaton counts the number of variables of the `VARIABLES` collection which take their value in `VALUES` and finally assigns this number to `NVAR`.

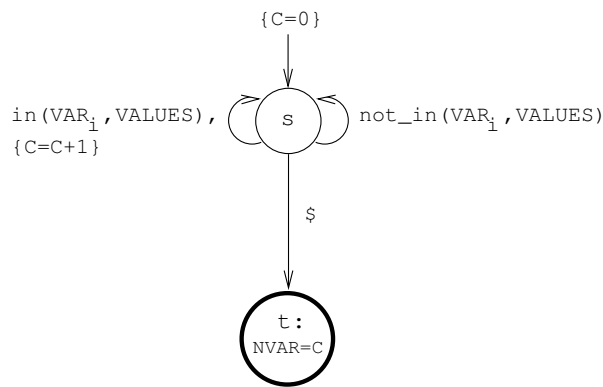


Figure 4.19: Automaton of the `among` constraint

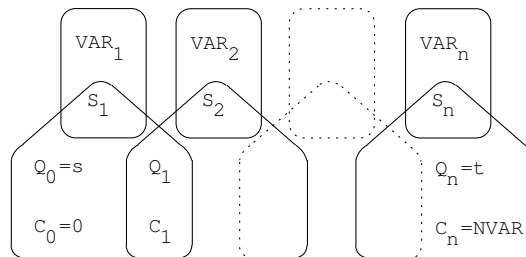


Figure 4.20: Hypergraph of the reformulation corresponding to the automaton of the `among` constraint

Remark

A similar constraint called `between` was introduced in CHIP in 1990.

The `common` constraint can be seen as a generalization of the `among` constraint where we allow the `val` attributes of the `VALUES` collection to be domain variables.

See also

`among_diff_0`, `exactly`, `global_cardinality`, `count`, `common`, `nvalue`, `max_nvalue`, `min_nvalue`.

Key words

value constraint, counting constraint, automaton, automaton with counters, alpha-acyclic constraint network(2).

4.13 among_diff_0

Origin	Used in the automaton of <i>nvalue</i> .
Constraint	<code>among_diff_0(NVAR, VARIABLES)</code>
Argument(s)	<pre>NVAR : dvar VARIABLES : collection(var - dvar)</pre>
Restriction(s)	<pre>NVAR ≥ 0 NVAR ≤ VARIABLES required(VARIABLES, var)</pre>
Purpose	NVAR is the number of variables of the collection VARIABLES which take a value different from 0.
<hr/>	
Arc input(s)	VARIABLES
Arc generator	<i>SELF</i> \mapsto <code>collection(variables)</code>
Arc arity	1
Arc constraint(s)	<code>variables.var ≠ 0</code>
Graph property(ies)	NARC = NVAR

Example $\text{among_diff_0} \left(3, \left\{ \begin{array}{l} \text{var} - 0, \\ \text{var} - 5, \\ \text{var} - 5, \\ \text{var} - 0, \\ \text{var} - 1 \end{array} \right\} \right)$

Parts (A) and (B) of Figure 4.21 respectively show the initial and final graph. Since we use the **NARC** graph property, the unary arcs of the final graph are stressed in bold.

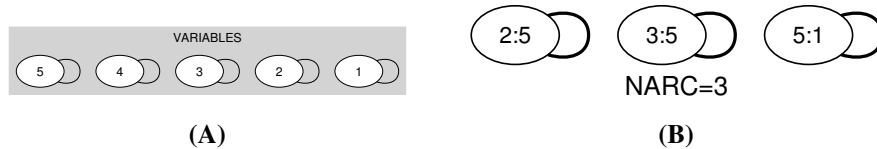


Figure 4.21: Initial and final graph of the `among_diff_0` constraint

Graph model Since this is a unary constraint we employ the *SELF* arc generator in order to produce an initial graph with a single loop on each vertex.

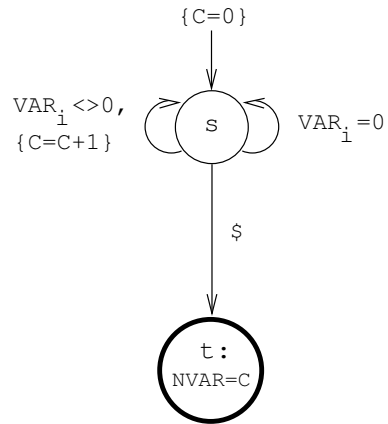


Figure 4.22: Automaton of the among_diff_0 constraint

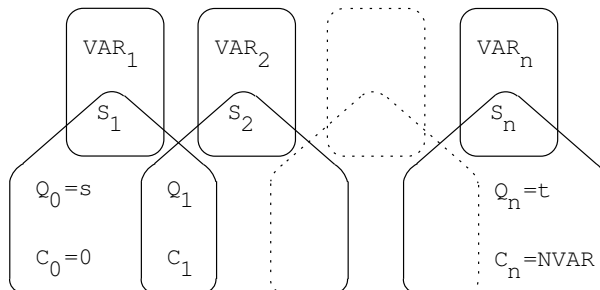


Figure 4.23: Hypergraph of the reformulation corresponding to the automaton of the among_diff_0 constraint

- Automaton** Figure 4.22 depicts the automaton associated to the `among_diff_0` constraint. To each variable VAR_i of the collection `VARIABLES` corresponds a 0-1 signature variable S_i . The following signature constraint links VAR_i and S_i : $VAR_i \neq 0 \Leftrightarrow S_i$. The automaton counts the number of variables of the `VARIABLES` collection which take a value different from 0 and finally assigns this number to `NVAR`.
- See also** `among`, `nvalue`.
- Key words** value constraint, counting constraint, joker value, automaton, automaton with counters, alpha-acyclic constraint network(2).

20040807

211

4.14 among_interval

Origin	Derived from among.
Constraint	among_interval(NVAR, VARIABLES, LOW, UP)
Argument(s)	NVAR : dvar VARIABLES : collection(var - dvar) LOW : int UP : int
Restriction(s)	NVAR ≥ 0 NVAR ≤ VARIABLES required(VARIABLES, var) LOW ≤ UP
Purpose	NVAR is the number of variables of the collection VARIABLES taking a value that is located within interval [LOW, UP].
Arc input(s)	VARIABLES
Arc generator	SELF ↦ collection(variables)
Arc arity	1
Arc constraint(s)	<ul style="list-style-type: none"> • LOW ≤ variables.var • variables.var ≤ UP
Graph property(ies)	NARC = NVAR

Example among_interval $\left(3, \left\{ \begin{array}{l} \text{var} - 4, \\ \text{var} - 5, \\ \text{var} - 8, \\ \text{var} - 4, \\ \text{var} - 1 \end{array} \right\}, 3, 5 \right)$

The constraint holds since we have 3 values, namely 4, 5 and 4 which are situated within interval [3, 5]. Parts (A) and (B) of Figure 4.24 respectively show the initial and final graph. Since we use the **NARC** graph property, the unary arcs of the final graph are stressed in bold.

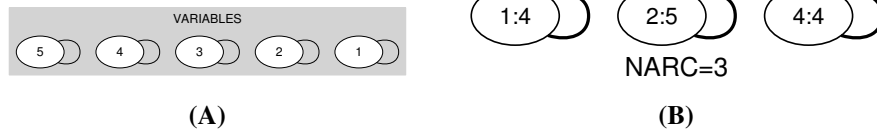


Figure 4.24: Initial and final graph of the among_interval constraint

Graph model

The arc constraint corresponds to a unary constraint. For this reason we employ the *SELF* arc generator in order to produce a graph with a single loop on each vertex.

Automaton

Figure 4.25 depicts the automaton associated to the *among_interval* constraint. To each variable VAR_i of the collection *VARIABLES* corresponds a 0-1 signature variable S_i . The following signature constraint links VAR_i and S_i : $LOW \leq VAR_i \wedge VAR_i \leq UP \Leftrightarrow S_i$. The automaton counts the number of variables of the *VARIABLES* collection which take their value in $[LOW, UP]$ and finally assigns this number to *NVAR*.

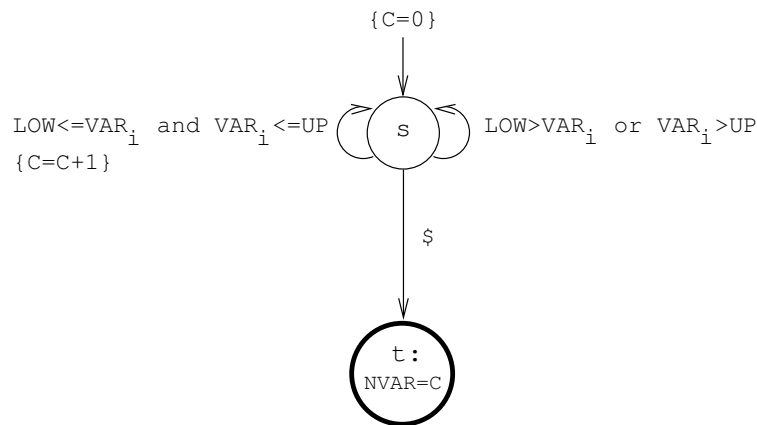


Figure 4.25: Automaton of the *among_interval* constraint

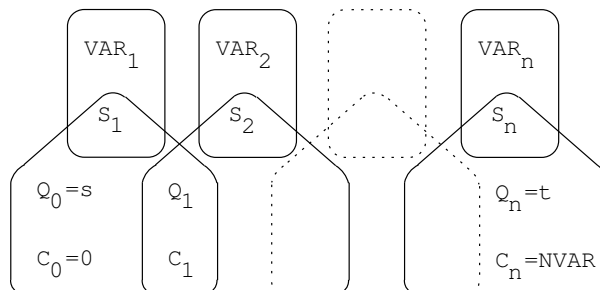


Figure 4.26: Hypergraph of the reformulation corresponding to the automaton of the *among_interval* constraint

Remark

By giving explicitly all values of the interval $[LOW, UP]$ the *among_interval* constraint can be modelled with the *among* constraint. However when $LOW - UP + 1$ is a large quantity the *among_interval* constraint provides a more compact form.

See also

among.

Key words

value constraint, counting constraint, interval, automaton, automaton with counters, alpha-acyclic constraint network(2).

4.15 among_low_up

Origin	[37]
Constraint	among_low_up(LOW, UP, VARIABLES, VALUES)
Argument(s)	LOW : int UP : int VARIABLES : collection(var - dvar) VALUES : collection(val - int)
Restriction(s)	LOW \geq 0 LOW \leq VARIABLES UP \geq LOW required(VARIABLES, var) required(VALUES, val) distinct(VALUES, val)
Purpose	Between LOW and UP variables of the VARIABLES collection are assigned to a value of the VALUES collection.
Arc input(s)	VARIABLES VALUES
Arc generator	<i>PRODUCT</i> \mapsto collection(variables, values)
Arc arity	2
Arc constraint(s)	variables.var = values.val
Graph property(ies)	<ul style="list-style-type: none"> • NARC \geq LOW • NARC \leq UP
Example	$\text{among_low_up} \left(1, 2, \left\{ \begin{array}{l} \text{var} - 9, \text{var} - 2, \text{var} - 4, \text{var} - 5, \\ \text{val} - 0, \\ \text{val} - 2, \\ \text{val} - 4, \\ \text{val} - 6, \\ \text{val} - 8 \end{array} \right\} \right)$ <p>Parts (A) and (B) of Figure 4.27 respectively show the initial and final graph. Since we use the NARC graph property, the arcs of the final graph are stressed in bold. The among_low_up constraint holds since between 1 and 2 variables of the VARIABLES collection are assigned to a value of the VALUES collection.</p>
Graph model	Each arc constraint of the final graph corresponds to the fact that a variable is assigned to a value that belong to the VALUES collection. The two graph properties restrict the total number of arcs to the interval [LOW, UP].

- Automaton** Figure 4.28 depicts the automaton associated to the `among_low_up` constraint. To each variable VAR_i of the collection `VARIABLES` corresponds a 0-1 signature variable S_i . The following signature constraint links VAR_i and S_i : $VAR_i \in VALUES \Leftrightarrow S_i$. The automaton counts the number of variables of the `VARIABLES` collection which take their value in `VALUES` and finally checks that this number is within the interval `[LOW, UP]`.
- Used in** `among_seq`, `cycle_card_on_path`, `interval_and_count`, `sliding_card_skip0`.
- See also** `among`.
- Key words** value constraint, counting constraint, automaton, automaton with counters, alpha-acyclic constraint network(2), acyclic, bipartite, no_loop.

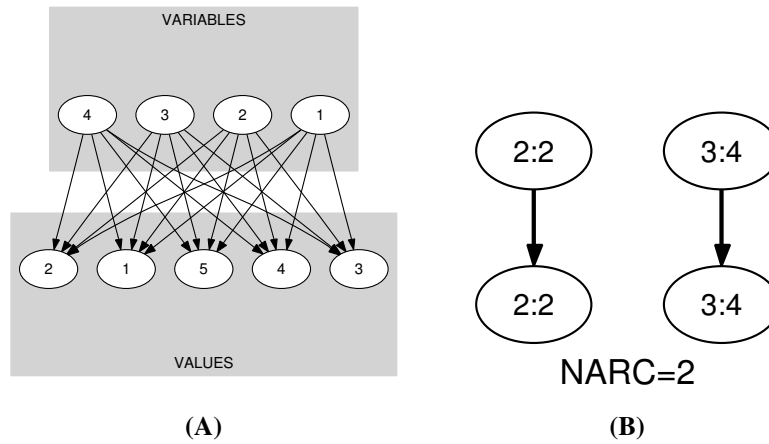


Figure 4.27: Initial and final graph of the among_low_up constraint

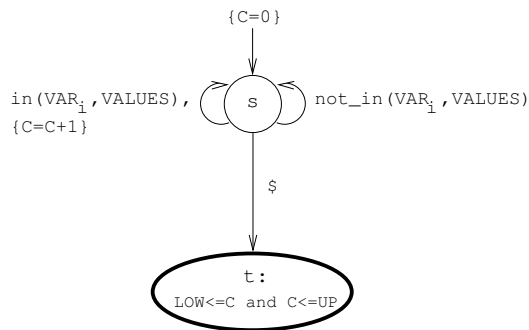


Figure 4.28: Automaton of the among_low_up constraint

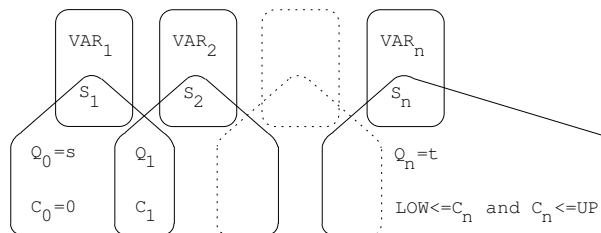


Figure 4.29: Hypergraph of the reformulation corresponding to the automaton of the among_low_up constraint

20030820

217

4.16 among_modulo

Origin	Derived from among.
Constraint	among_modulo(NVAR, VARIABLES, REMAINDER, QUOTIENT)
Argument(s)	NVAR : dvar VARIABLES : collection(var - dvar) REMAINDER : int QUOTIENT : int
Restriction(s)	NVAR ≥ 0 NVAR $\leq \text{VARIABLES} $ required(VARIABLES, var) REMAINDER ≥ 0 REMAINDER < QUOTIENT QUOTIENT > 0
Purpose	NVAR is the number of variables of the collection VARIABLES taking a value that is congruent to REMAINDER modulo QUOTIENT.
Arc input(s)	VARIABLES
Arc generator	SELF \mapsto collection(variables)
Arc arity	1
Arc constraint(s)	variables.var mod QUOTIENT = REMAINDER
Graph property(ies)	NARC = NVAR
Example	$\text{among_modulo} \left(3, \left\{ \begin{array}{l} \text{var} - 4, \\ \text{var} - 5, \\ \text{var} - 8, \\ \text{var} - 4, \\ \text{var} - 1 \end{array} \right\}, 0, 2 \right)$ <p>In this example REMAINDER = 0 and QUOTIENT = 2 specifies that we count the number of even values taken by the different variables. Parts (A) and (B) of Figure 4.30 respectively show the initial and final graph. Since we use the NARC graph property, the unary arcs of the final graph are stressed in bold.</p>
Graph model	The arc constraint corresponds to a unary constraint. For this reason we employ the <i>SELF</i> arc generator in order to produce a graph with a single loop on each vertex.
Automaton	Figure 4.31 depicts the automaton associated to the among_modulo constraint. To each variable VAR _i of the collection VARIABLES corresponds a 0-1 signature variable S _i . The following signature constraint links VAR _i and S _i : VAR _i mod QUOTIENT = REMAINDER \Leftrightarrow S _i .

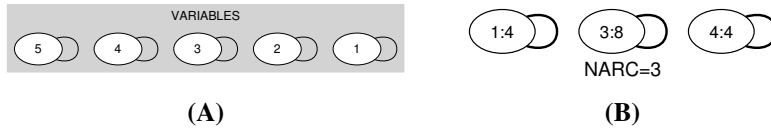


Figure 4.30: Initial and final graph of the among_modulo constraint

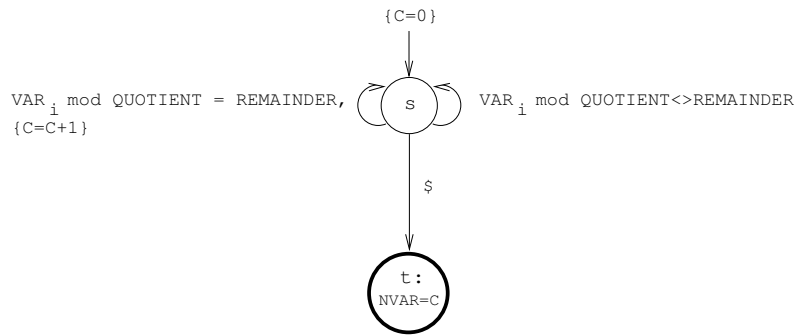


Figure 4.31: Automaton of the among_modulo constraint

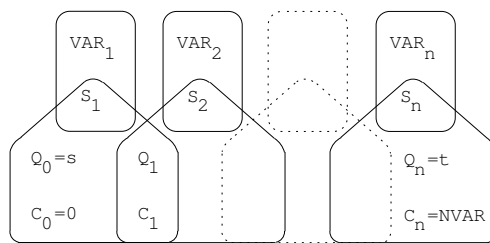


Figure 4.32: Hypergraph of the reformulation corresponding to the automaton of the among_modulo constraint

- Remark** By giving explicitly all values v which satisfy the equality $v \bmod \text{QUOTIENT} = \text{REMAINDER}$ the `among_modulo` constraint can be modelled with the `among` constraint. However the `among_modulo` constraint provides a more compact form.
- See also** `among`.
- Key words** value constraint, counting constraint, modulo, automaton, automaton with counters, alpha-acyclic constraint network(2).

20030820

221

4.17 among_seq

Origin	[37]
Constraint	among_seq(LOW, UP, SEQ, VARIABLES, VALUES)
Argument(s)	LOW : int UP : int SEQ : int VARIABLES : collection(var – dvar) VALUES : collection(val – int)
Restriction(s)	LOW \geq 0 LOW \leq VARIABLES UP \geq LOW SEQ $>$ 0 SEQ \geq LOW SEQ \leq VARIABLES required(VARIABLES, var) required(VALUES, val) distinct(VALUES, val)
Purpose	Constrains all sequences of SEQ consecutive variables of the collection VARIABLES to take at least LOW values in VALUES and at most UP values in VALUES.
Arc input(s)	VARIABLES
Arc generator	PATH \mapsto collection
Arc arity	SEQ
Arc constraint(s)	among_low_up(LOW, UP, collection, VALUES)
Graph property(ies)	$\text{NARC} = \text{VARIABLES} - \text{SEQ} + 1$

Example	$\text{among_seq} \left(1, 2, 4, \left\{ \begin{array}{l} \text{var} - 9, \\ \text{var} - 2, \\ \text{var} - 4, \\ \text{var} - 5, \\ \text{var} - 5, \\ \text{var} - 7, \\ \text{var} - 2 \end{array} \right\}, \left\{ \begin{array}{l} \text{val} - 0, \\ \text{val} - 2, \\ \text{val} - 4, \\ \text{val} - 6, \\ \text{val} - 8 \end{array} \right\} \right)$
----------------	--

The previous constraint holds since the different sequences of 4 consecutive variables contains respectively 2, 2, 1 and 1 even numbers.

Graph model	A constraint on sliding sequences of consecutive variables. Each vertex of the graph corresponds to a variable. Since they link SEQ variables, the arcs of the graph correspond to hyperarcs. In order to link SEQ consecutive variables we use the arc generator <i>PATH</i> . The constraint associated to an arc corresponds to the <code>among_low_up</code> constraint defined at another entry of this catalog.
Signature	Since we use the <i>PATH</i> arc generator with an arity of SEQ on the items of the <i>VARIABLES</i> collection, the expression $ VARIABLES - SEQ + 1$ corresponds to the maximum number of arcs of the final graph. Therefore we can rewrite the graph property $NARC = \frac{ VARIABLES - SEQ + 1}{NARC}$ to $NARC \geq VARIABLES - SEQ + 1$ and simplify <u>NARC</u> to <u>NARC</u> .
Algorithm	[65].
See also	<code>among</code> , <code>among_low_up</code> .
Key words	decomposition, sliding sequence constraint, sequence, hypergraph.

4.18 arith

Origin	Used in the definition of several automata
Constraint	$\text{arith}(\text{VARIABLES}, \text{RELOP}, \text{VALUE})$
Argument(s)	<p>VARIABLES : collection(var – dvar)</p> <p>RELOP : atom</p> <p>VALUE : int</p>
Restriction(s)	<p>required(VARIABLES, var)</p> <p>RELOP $\in \{=, \neq, <, \geq, >, \leq\}$</p>
Purpose	Enforce for all variables var of the VARIABLES collection to have var RELOP VALUE.
Arc input(s)	VARIABLES
Arc generator	$\text{SELF} \mapsto \text{collection}(\text{variables})$
Arc arity	1
Arc constraint(s)	variables.var RELOP VALUE
Graph property(ies)	$\text{NARC} = \text{VARIABLES} $

Example $\text{arith} \left(\left(\begin{array}{c} \text{var} - 4, \\ \text{var} - 5, \\ \text{var} - 7, \\ \text{var} - 4, \\ \text{var} - 5 \end{array} \right), <, 9 \right)$

The constraint holds since all variables of are stricly less than 9. Parts (A) and (B) of Figure 4.33 respectively show the initial and final graph. Since we use the **NARC** graph property, the unary arcs of the final graph are stressed in bold.



Figure 4.33: Initial and final graph of the `arith` constraint

Automaton Figure 4.34 depicts the automaton associated to the `arith` constraint. To each variable VAR_i of the collection `VARIABLES` corresponds a 0-1 signature variable S_i . The following signature constraint links VAR_i and S_i : $\text{VAR}_i \text{ RELOP VALUE} \Leftrightarrow S_i$. The automaton enforces for each variable VAR_i the condition $\text{VAR}_i \text{ RELOP VALUE}$.

Used in `arith_sliding`.

See also among, count.

Key words decomposition, value constraint, domain definition, automaton,
automaton without counters.

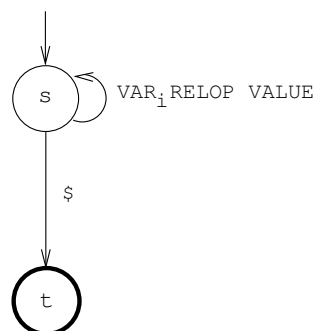


Figure 4.34: Automaton of the arith constraint

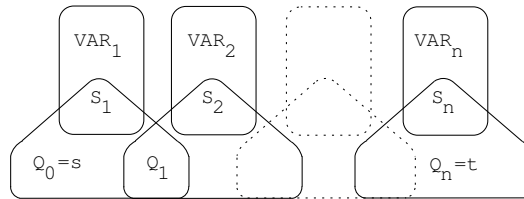


Figure 4.35: Hypergraph of the reformulation corresponding to the automaton of the arith constraint

20040814

227

4.19 arith_or

Origin	Used in the definition of several automata
Constraint	<code>arith_or(VARIABLES1, VARIABLES2, RELOP, VALUE)</code>
Argument(s)	<code>VARIABLES1 : collection(var - dvar)</code> <code>VARIABLES2 : collection(var - dvar)</code> <code>RELOP : atom</code> <code>VALUE : int</code>
Restriction(s)	<code>required(VARIABLES1, var)</code> <code>required(VARIABLES2, var)</code> <code> VARIABLES1 = VARIABLES2 </code> <code>RELOP ∈ [=, ≠, <, ≥, >, ≤]</code>
Purpose	Enforce for all pairs of variables $\text{var1}_i, \text{var2}_i$ of the <code>VARIABLES1</code> and <code>VARIABLES2</code> collections to have $\text{var1}_i \text{ RELOP VALUE} \vee \text{var2}_i \text{ RELOP VALUE}$.
Arc input(s)	<code>VARIABLES1 VARIABLES2</code>
Arc generator	<code>PRODUCT(=) ↦ collection(variables1, variables2)</code>
Arc arity	2
Arc constraint(s)	<code>variables1.var RELOP VALUE ∨ variables2.var RELOP VALUE</code>
Graph property(ies)	<code>NARC = VARIABLES1 </code>

Example	$\text{arith_or} \left(\left(\left\{ \begin{array}{l} \text{var} - 0, \\ \text{var} - 1, \\ \text{var} - 0, \\ \text{var} - 0, \\ \text{var} - 1 \\ \text{var} - 0, \\ \text{var} - 0, \\ \text{var} - 0, \\ \text{var} - 1, \\ \text{var} - 0 \end{array} \right\}, \right. \right. \\ \left. \left. \left\{ \begin{array}{l} \text{var} - 0, \\ \text{var} - 1, \\ \text{var} - 0 \end{array} \right\}, =, 0 \right)$
----------------	--

The constraint holds since for all pairs of variables $\text{var1}_i, \text{var2}_i$ of the `VARIABLES1` and `VARIABLES2` collections we have that at least one of the variables is equal to 0. Parts (A) and (B) of Figure 4.36 respectively show the initial and final graphs. Since we use the **NARC** graph property, the unary arcs of the final graph are stressed in bold.

Automaton Figure 4.37 depicts the automaton associated to the `arith_or` constraint. Let VAR1_i and VAR2_i be the i^{th} variables of the `VARIABLES1` and `VARIABLES2` collections. To each pair of variables $(\text{VAR1}_i, \text{VAR2}_i)$ corresponds a signature variable S_i . The following signature constraint links $\text{VAR1}_i, \text{VAR2}_i$ and S_i : $\text{VAR1}_i \text{ RELOP VALUE} \vee \text{VAR2}_i \text{ RELOP VALUE} \Leftrightarrow S_i$. The automaton enforces for each pair of variables $\text{VAR1}_i, \text{VAR2}_i$ the condition $\text{VAR1}_i \text{ RELOP VALUE} \vee \text{VAR2}_i \text{ RELOP VALUE}$.

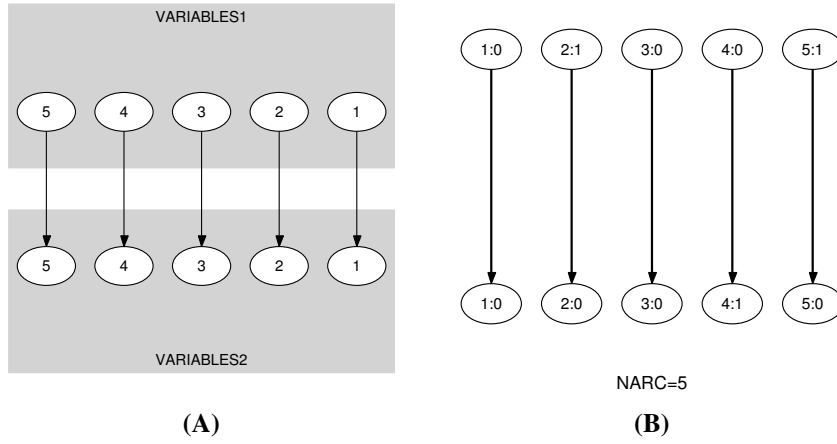


Figure 4.36: Initial and final graph of the `arith_or` constraint

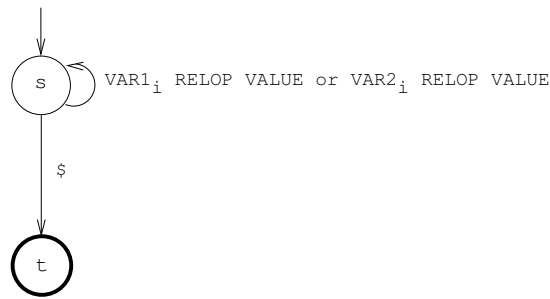


Figure 4.37: Automaton of the `arith_or` constraint

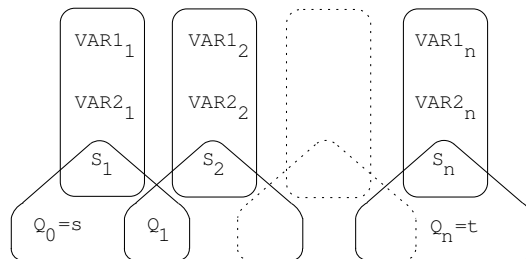


Figure 4.38: Hypergraph of the reformulation corresponding to the automaton of the `arith_or` constraint

See also arith.

Key words decomposition, value constraint, automaton, automaton without counters, acyclic, bipartite, no_loop.

20040814

231

4.20 arith_sliding

Origin	Used in the definition of some automaton
Constraint	<code>arith_sliding(VARIABLES, RELOP, VALUE)</code>
Argument(s)	<code>VARIABLES</code> : <code>collection(var - dvar)</code> <code>RELOP</code> : <code>atom</code> <code>VALUE</code> : <code>int</code>
Restriction(s)	<code>required(VARIABLES, var)</code> <code>RELOP</code> \in $\{=, \neq, <, \geq, >, \leq\}$
Purpose	Enforce for all sequences of variables $\text{var}_1, \text{var}_2, \dots, \text{var}_i$ of the <code>VARIABLES</code> collection to have $(\text{var}_1 + \text{var}_2 + \dots + \text{var}_i)$ <code>RELOP VALUE</code> .
Arc input(s)	<code>VARIABLES</code>
Arc generator	<code>PATH_1</code> \mapsto <code>collection</code>
Arc arity	*
Arc constraint(s)	<code>arith(collection, RELOP, VALUE)</code>
Graph property(ies)	<code>NARC</code> = $ \text{VARIABLES} $
Example	$\text{arith_sliding} \left(\left(\begin{array}{c} \text{var} - 0, \\ \text{var} - 0, \\ \text{var} - 1, \\ \text{var} - 2, \\ \text{var} - 0, \\ \text{var} - 0, \\ \text{var} - -3 \end{array} \right) , <, 4 \right)$ <p>The previous constraint holds since all the following seven inequalities hold:</p> <ul style="list-style-type: none"> • $0 < 4$, • $0 + 0 < 4$, • $0 + 0 + 1 < 4$, • $0 + 0 + 1 + 2 < 4$, • $0 + 0 + 1 + 2 + 0 < 4$, • $0 + 0 + 1 + 2 + 0 + 0 < 4$, • $0 + 0 + 1 + 2 + 0 + 0 - 3 < 4$.
Automaton	Figure 4.39 depicts the automaton associated to the <code>arith_sliding</code> constraint. To each item of the collection <code>VARIABLES</code> corresponds a signature variable S_i , which is equal to 0.
See also	<code>arith</code> , <code>cumulative</code> .
Key words	decomposition, sliding sequence constraint, sequence, hypergraph, automaton, automaton with counters.

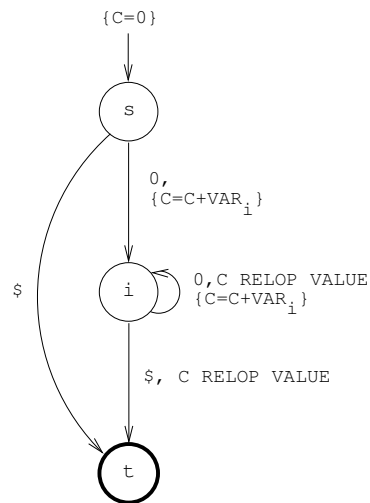


Figure 4.39: Automaton of the `arith_sliding` constraint

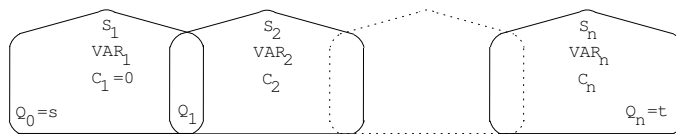


Figure 4.40: Hypergraph of the reformulation corresponding to the automaton of the `arith_sliding` constraint

4.21 assign_and_counts

Origin	N. Beldiceanu
Constraint	assign_and_counts(COLOURS, ITEMS, RELOP, LIMIT)
Argument(s)	COLOURS : collection(val - int) ITEMS : collection(bin - dvar, colour - dvar) RELOP : atom LIMIT : dvar
Restriction(s)	required(COLOURS, val) distinct(COLOURS, val) required(ITEMS, [bin, colour]) RELOP \in [=, \neq , <, \geq , >, \leq]
Purpose	Given several items (each of them having a specific colour which may not be initially fixed), and different bins, assign each item to a bin, so that the total number n of items of colour COLOURS in each bin satisfies the condition n RELOP LIMIT.
Derived Collection(s)	col(VALUEs - collection(val - int), [item(val - COLOURS.val)])
Arc input(s)	ITEMS ITEMS
Arc generator	PRODUCT \mapsto collection(items1, items2)
Arc arity	2
Arc constraint(s)	items1.bin = items2.bin
Sets	SUCC \mapsto $\left[\begin{array}{l} \text{source,} \\ \text{variables - col} \left(\begin{array}{l} \text{VARIABLES - collection(var - dvar),} \\ \text{[item(var - ITEMS.colour)]} \end{array} \right) \end{array} \right]$
Constraint(s) on sets	counts(VALUEs, variables, RELOP, LIMIT)
Example	$\text{assign_and_counts} \left(\begin{array}{l} \{ \text{val} - 4 \}, \\ \left\{ \begin{array}{l} \text{bin} - 1 \quad \text{colour} - 4, \\ \text{bin} - 3 \quad \text{colour} - 4, \\ \text{bin} - 1 \quad \text{colour} - 4, \\ \text{bin} - 1 \quad \text{colour} - 5 \end{array} \right\}, \leq, 2 \end{array} \right)$

Parts (A) and (B) of Figure 4.41 respectively show the initial and final graph. The final graph consists of the following two connected components:

- The connected component containing six vertices corresponds to the items which are assigned to bin 1.

- The connected component containing two vertices corresponds to the items which are assigned to bin 3.

The `assign_and_counts` constraint holds since for each set of successors of the vertices of the final graph no more than two items take colour 4. Figure 4.42 shows the solution associated to the example. The items and the bins are respectively represented by little squares and by the different columns. Each little square contains the value of the key attribute of the item to which it corresponds. The items for which the colour attribute is equal to 4 are located under the thick line.

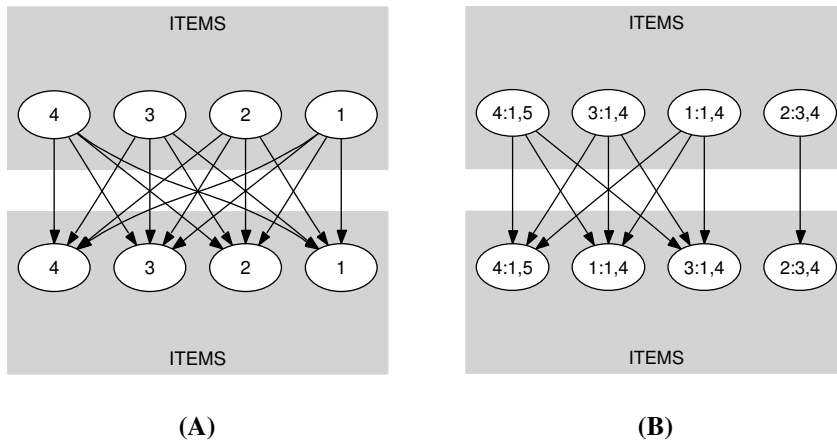


Figure 4.41: Initial and final graph of the `assign_and_counts` constraint

<>4					
	4				
	3				<3
=4	1		2		
	1	2	3	4	5

Figure 4.42: Assignment of the items to the bins

Graph model

We enforce the counts constraint on the colour of the items that are assigned to the same bin.

Automaton

Figure 4.43 depicts the automaton associated to the `assign_and_counts` constraint. To each `colour` attribute $COLOUR_i$ of the collection `ITEMS` corresponds a 0-1 signature variable S_i . The following signature constraint links $COLOUR_i$ and S_i : $COLOUR_i \in COLOURS \Leftrightarrow S_i$. For all items of the collection `ITEMS` for which the `colour` attribute takes its value in `COLOURS`, counts for each value assigned to the bin attribute its number of occurrences n , and finally imposes the condition $n \leq RELOP \text{ LIMIT}$.

Usage	Some persons have pointed out that it is impossible to use constraints such as <code>among</code> , <code>atleast</code> , <code>atmost</code> , <code>count</code> , or <code>global_cardinality</code> if the set of variables is not initially known. However, this is for instance required in practice for some timetabling problems.
See also	<code>count</code> , <code>counts</code> .
Key words	assignment, coloured, automaton, automaton with array of counters, derived collection.

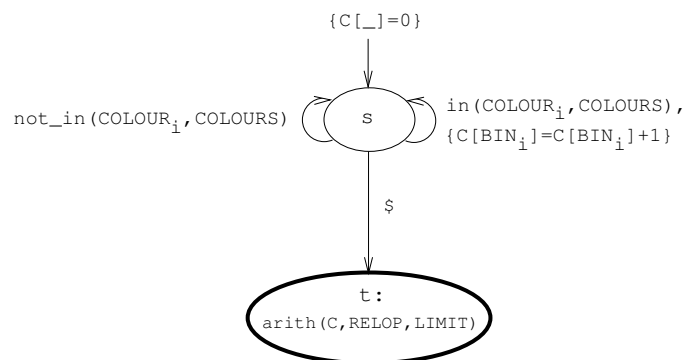


Figure 4.43: Automaton of the `assign_and_counts` constraint

20000128

237

4.22 assign_and_nvalues

Origin	Derived from <code>assign_and_counts</code> and <code>nvalues</code> .
Constraint	<code>assign_and_nvalues</code> (ITEMS, RELOP, LIMIT)
Argument(s)	ITEMS : <code>collection(bin - dvar, value - dvar)</code> RELOP : <code>atom</code> LIMIT : <code>dvar</code>
Restriction(s)	<code>required</code> (ITEMS, [bin, value]) RELOP \in [=, \neq , <, \geq , >, \leq]
Purpose	Given several items (each of them having a specific value which may not be initially fixed), and different bins, assign each item to a bin, so that the number n of distinct values in each bin satisfies the condition n RELOP LIMIT.
Arc input(s)	ITEMS ITEMS
Arc generator	$PRODUCT \mapsto \text{collection}(\text{items1}, \text{items2})$
Arc arity	2
Arc constraint(s)	<code>items1.bin = items2.bin</code>
Sets	$SUCC \mapsto$ $\left[\begin{array}{l} \text{source,} \\ \text{variables} - \text{col}(\text{VARIABLES} - \text{collection}(\text{var} - \text{dvar}), [\text{item}(\text{var} - \text{ITEMS.value})]) \end{array} \right]$
Constraint(s) on sets	<code>nvalues</code> (variables, RELOP, LIMIT)

Example $\text{assign_and_nvalues} \left(\left(\left(\begin{array}{ll} \text{bin} - 2 & \text{value} - 3, \\ \text{bin} - 1 & \text{value} - 5, \\ \text{bin} - 2 & \text{value} - 3, \\ \text{bin} - 2 & \text{value} - 3, \\ \text{bin} - 2 & \text{value} - 4 \end{array} \right), \leq, 2 \right) \right)$

Parts (A) and (B) of Figure 4.44 respectively show the initial and final graph. The final graph consists of the following two connected components:

- The connected component containing eight vertices corresponds to the items which are assigned to bin 2.
- The connected component containing two vertices corresponds to the items which are assigned to bin 1.

The `assign_and_nvalues` constraint holds since for each set of successors of the vertices of the final graph no more than two distinct values are used:

- The unique item assigned to bin 1 uses value 5.

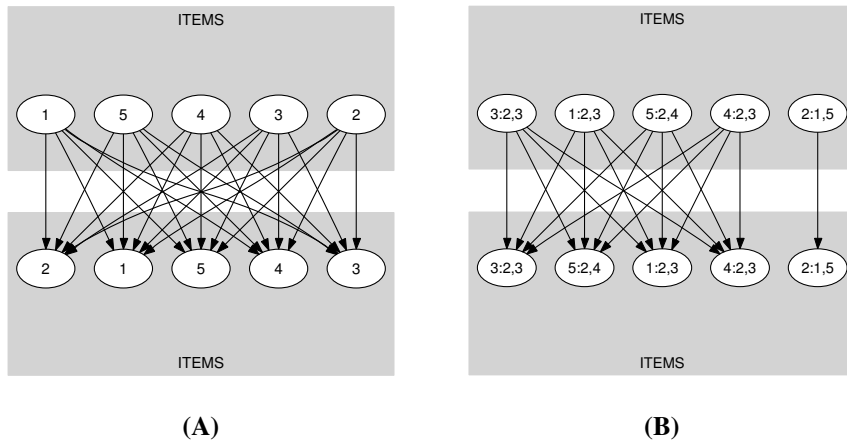


Figure 4.44: Initial and final graph of the `assign_and_nvalues` constraint

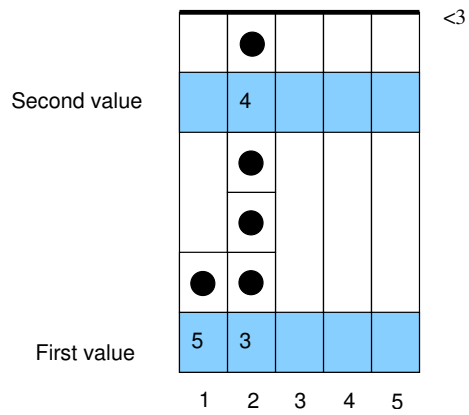


Figure 4.45: An assignment with at most two distinct values in parallel

- Items assigned to bin 2 use values 3 and 4.

Figure 4.45 depicts the solution corresponding to the example.

Graph model

We enforce the `nvalue` constraint on the items that are assigned to the same bin.

Usage

Let us give two examples where the `assign_and_nvalues` constraint is useful:

- Quite often, in bin-packing problems, each item has a specific type, and one wants to assign items of similar type to each bin.
- In a vehicle routing problem, one wants to restrict the number of towns visited by each vehicle. Note that several customers may be located at the same town. In this example, each bin would correspond to a vehicle, each item would correspond to a visit to a customer, and the colour of an item would be the location of the corresponding customer.

See also

`nvalue`, `nvalues`.

Key words

assignment, number of distinct values.

4.23 atleast

Origin	CHIP
Constraint	<code>atleast(N, VARIABLES, VALUE)</code>
Argument(s)	<code>N</code> : int <code>VARIABLES</code> : <code>collection(var - dvar)</code> <code>VALUE</code> : int
Restriction(s)	$N \geq 0$ $N \leq \text{VARIABLES} $ <code>required(VARIABLES, var)</code>
Purpose	At least N variables of the VARIABLES collection are assigned to value VALUE.
Arc input(s)	VARIABLES
Arc generator	$\text{SELF} \mapsto \text{collection}(\text{variables})$
Arc arity	1
Arc constraint(s)	<code>variables.var = VALUE</code>
Graph property(ies)	$\text{NARC} \geq N$
Example	<code>atleast(2, {var - 4, var - 2, var - 4, var - 5}, 4)</code>

Parts (A) and (B) of Figure 4.46 respectively show the initial and final graph. Since we use the **NARC** graph property, the unary arcs of the final graph are stressed in bold. The `atleast` constraint holds since at least 2 variables are assigned to value 4.

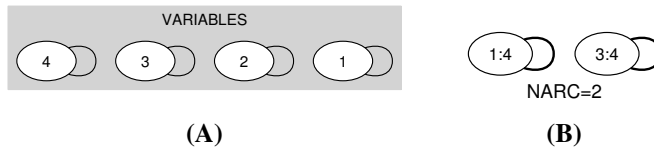


Figure 4.46: Initial and final graph of the `atleast` constraint

Graph model	Since we use a unary arc constraint (<code>VALUE</code> is fixed) we employ the <i>SELF</i> arc generator in order to produce a graph with a single loop on each vertex.
Automaton	Figure 4.47 depicts the automaton associated to the <code>atleast</code> constraint. To each variable VAR_i of the collection <code>VARIABLES</code> corresponds a 0-1 signature variable S_i . The following signature constraint links VAR_i and S_i : $\text{VAR}_i = \text{VALUE} \Leftrightarrow S_i$. The automaton counts the number of variables of the <code>VARIABLES</code> collection which are assigned to <code>VALUE</code> and finally checks that this number is greater than or equal to <code>N</code> .

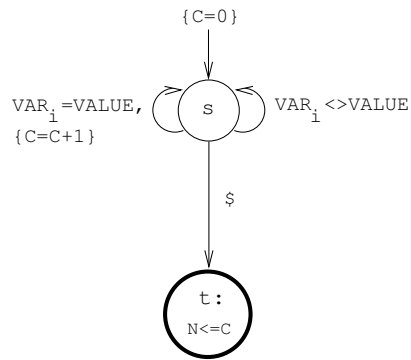


Figure 4.47: Automaton of the atLeast constraint

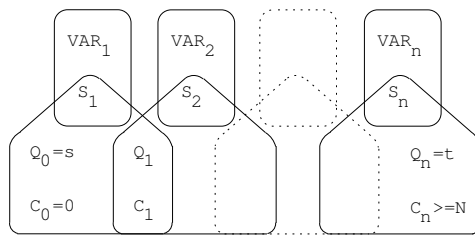


Figure 4.48: Hypergraph of the reformulation corresponding to the automaton of the atLeast constraint

See also atmost, among, exactly.

Key words value constraint, at least, automaton, automaton with counters,
alpha-acyclic constraint network(2).

20030820

245

4.24 atmost

Origin	CHIP
Constraint	<code>atmost(N, VARIABLES, VALUE)</code>
Argument(s)	<code>N</code> : int <code>VARIABLES</code> : collection(var – dvar) <code>VALUE</code> : int
Restriction(s)	$N \geq 0$ <code>required(VARIABLES, var)</code>
Purpose	At most N variables of the VARIABLES collection are assigned to value VALUE.
Arc input(s)	VARIABLES
Arc generator	<i>SELF</i> \mapsto collection(variables)
Arc arity	1
Arc constraint(s)	<code>variables.var = VALUE</code>
Graph property(ies)	$\mathbf{NARC} \leq N$
Example	<code>atmost(1, {var – 4, var – 2, var – 4, var – 5}, 2)</code>

Parts (A) and (B) of Figure 4.49 respectively show the initial and final graph. Since we use the **NARC** graph property, the unary arcs of the final graph are stressed in bold. The `atmost` constraint holds since at most one variable is assigned to value 2.

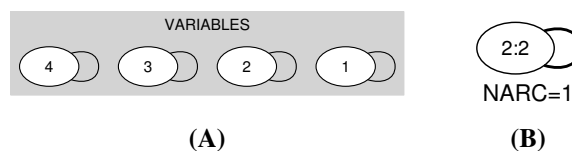


Figure 4.49: Initial and final graph of the `atmost` constraint

Graph model	Since we use a unary arc constraint (<code>VALUE</code> is fixed) we employ the <i>SELF</i> arc generator in order to produce a graph with a single loop on each vertex.
Automaton	Figure 4.50 depicts the automaton associated to the <code>atmost</code> constraint. To each variable VAR_i of the collection <code>VARIABLES</code> corresponds a 0-1 signature variable S_i . The following signature constraint links VAR_i and S_i : $\text{VAR}_i = \text{VALUE} \Leftrightarrow S_i$. The automaton counts the number of variables of the <code>VARIABLES</code> collection which are assigned to <code>VALUE</code> and finally checks that this number is less than or equal to <code>N</code> .

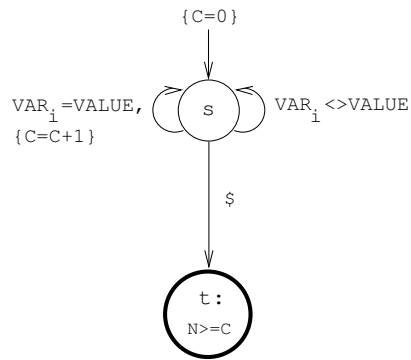


Figure 4.50: Automaton of the atleast constraint

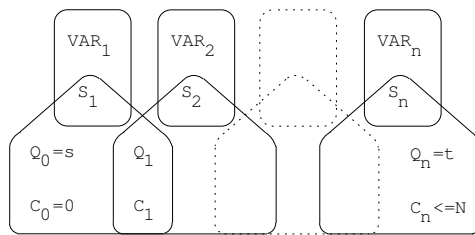


Figure 4.51: Hypergraph of the reformulation corresponding to the automaton of the atleast constraint

See also at least, among, exactly, cumulative.

Key words value constraint, at most, automaton, automaton with counters,
alpha-acyclic constraint network(2).

20030820

249

4.25 balance

Origin	N. Beldiceanu
Constraint	<code>balance(BALANCE, VARIABLES)</code>
Argument(s)	<code>BALANCE</code> : dvar <code>VARIABLES</code> : <code>collection(var - dvar)</code>
Restriction(s)	<code>BALANCE</code> ≥ 0 <code>BALANCE</code> $\leq \text{VARIABLES} $ <code>required(VARIABLES, var)</code>
Purpose	<p><code>BALANCE</code> is equal to the difference between the number of occurrence of the value that occurs the most and the value that occurs the least within the collection of variables <code>VARIABLES</code>.</p>
Arc input(s)	<code>VARIABLES</code>
Arc generator	<code>CLIQUE</code> \mapsto <code>collection(variables1, variables2)</code>
Arc arity	2
Arc constraint(s)	<code>variables1.var = variables2.var</code>
Graph property(ies)	<code>RANGE_NSCC = BALANCE</code>
Example	$\text{balance} \left(2, \left\{ \begin{array}{l} \text{var} - 3, \\ \text{var} - 1, \\ \text{var} - 7, \\ \text{var} - 1, \\ \text{var} - 1 \end{array} \right\} \right)$ <p>In this example, values 1, 3 and 7 are respectively used 3, 1 and 1 times. <code>BALANCE</code> is assigned to the difference between the maximum and minimum number of the previous occurrences (i.e. $3 - 1$). Parts (A) and (B) of Figure 4.52 respectively show the initial and final graph. Since we use the <code>RANGE_NSCC</code> graph property, we show the largest and smallest strongly connected components of the final graph.</p>
Graph model	The graph property <code>RANGE_NSCC</code> constrains the difference between the sizes of the largest and smallest strongly connected components.
Automaton	Figure 4.53 depicts the automaton associated to the <code>balance</code> constraint. To each item of the collection <code>VARIABLES</code> corresponds a signature variable S_i , which is equal to 1.
Usage	One application of this constraint is to enforce a <i>balanced assignment</i> of values, no matter how many distinct values will be used. In this case one will <i>push down</i> the maximum value of the first argument of the <code>balance</code> constraint.
See also	<code>balance_interval</code> , <code>balance_modulo</code> , <code>balance_partition</code> , <code>tree_range</code> .
Key words	value constraint, assignment, balanced assignment, automaton, automaton with array of counters, equivalence.

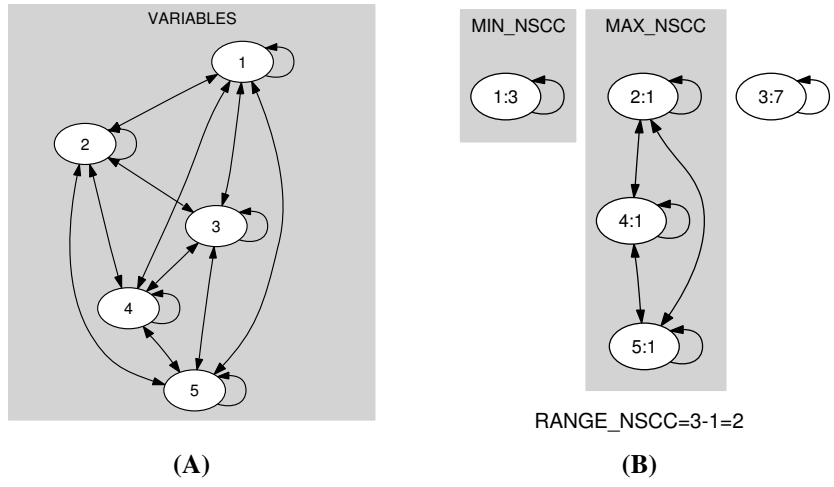


Figure 4.52: Initial and final graph of the balance constraint

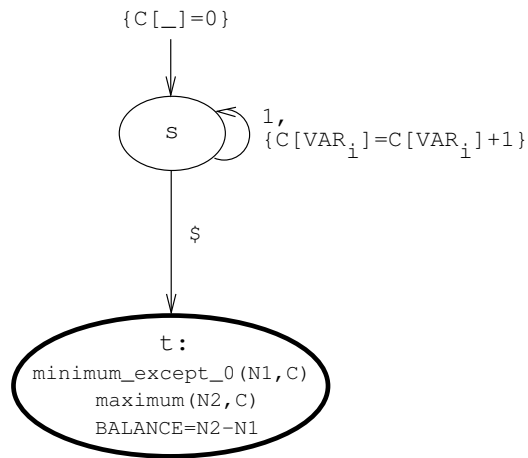


Figure 4.53: Automaton of the balance constraint

4.26 balance_interval

Origin	Derived from <code>balance</code> .
Constraint	<code>balance_interval(BALANCE, VARIABLES, SIZE_INTERVAL)</code>
Argument(s)	<code>BALANCE</code> : dvar <code>VARIABLES</code> : <code>collection(var - dvar)</code> <code>SIZE_INTERVAL</code> : int
Restriction(s)	<code>BALANCE</code> ≥ 0 <code>BALANCE</code> $\leq VARIABLES $ <code>required(VARIABLES, var)</code> <code>SIZE_INTERVAL</code> > 0
Purpose	<p>Consider the largest set \mathcal{S}_1 (respectively the smallest set \mathcal{S}_2) of variables of the collection <code>VARIABLES</code> which take their value in a same interval $[SIZE_INTERVAL \cdot k, SIZE_INTERVAL \cdot k + SIZE_INTERVAL - 1]$, where k is an integer. <code>BALANCE</code> is equal to the difference between the cardinality of \mathcal{S}_2 and the cardinality of \mathcal{S}_1.</p>
Arc input(s)	<code>VARIABLES</code>
Arc generator	<code>CLIQUE</code> \mapsto <code>collection(variables1, variables2)</code>
Arc arity	2
Arc constraint(s)	<code>variables1.var/SIZE_INTERVAL = variables2.var/SIZE_INTERVAL</code>
Graph property(ies)	<code>RANGE_NSCC = BALANCE</code>

Example `balance_interval` $\left(3, \left\{ \begin{array}{l} \text{var} - 6, \\ \text{var} - 4, \\ \text{var} - 3, \\ \text{var} - 3, \\ \text{var} - 4 \end{array} \right\}, 3 \right)$

In the previous example, the third parameter `SIZE_INTERVAL` defines the following family of intervals $[3 \cdot k, 3 \cdot k + 2]$, where k is an integer. Values 6,4,3,3 and 4 are respectively located within intervals $[6, 8]$, $[3, 5]$, $[3, 5]$, $[3, 5]$ and $[3, 5]$. Therefore intervals $[6, 8]$ and $[3, 5]$ are respectively used 1 and 4 times. `BALANCE` is assigned to the difference between the maximum and minimum number of the previous occurrences (i.e. $4 - 1$). Parts (A) and (B) of Figure 4.54 respectively show the initial and final graph. Since we use the `RANGE_NSCC` graph property, we show the largest and smallest strongly connected components of the final graph.

Graph model The graph property `RANGE_NSCC` constraints the difference between the sizes of the largest and smallest strongly connected components.

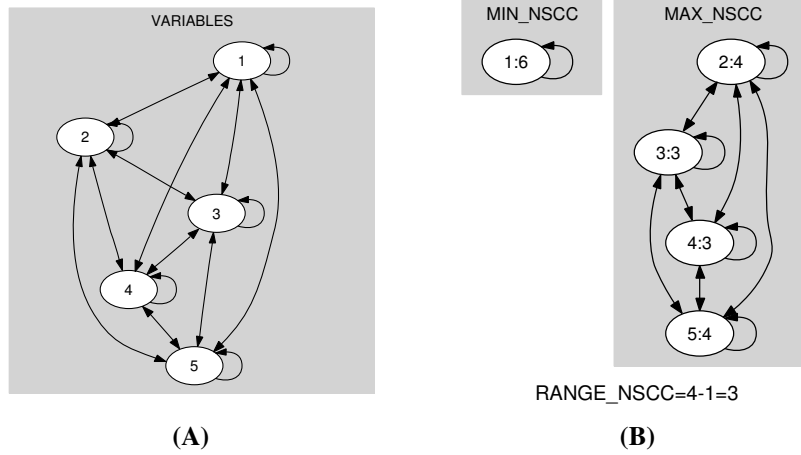


Figure 4.54: Initial and final graph of the balance_interval constraint

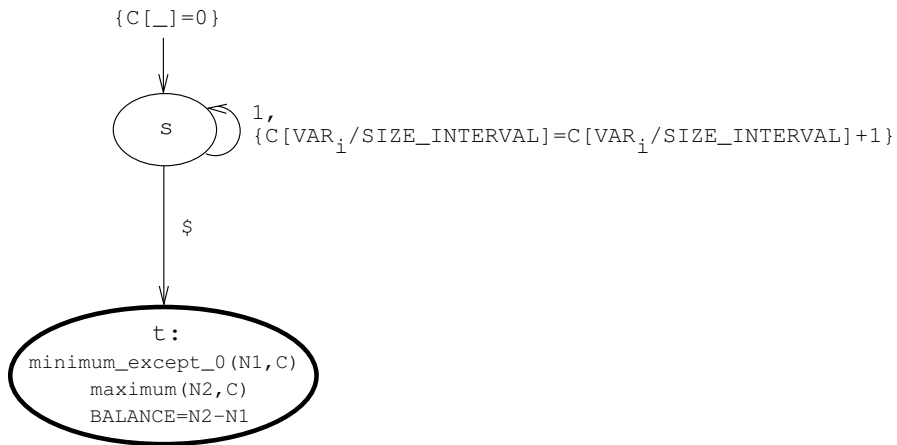


Figure 4.55: Automaton of the balance_interval constraint

- Automaton** Figure 4.55 depicts the automaton associated to the `balance_interval` constraint. To each item of the collection `VARIABLES` corresponds a signature variable S_i , which is equal to 1.
- Usage** One application of this constraint is to enforce a *balanced assignment* of interval of values, no matter how many distinct interval of values will be used. In this case one will *push down* the maximum value of the first argument of the `balance_interval` constraint.
- See also** `balance`.
- Key words** value constraint, interval, assignment, balanced assignment, automaton, automaton with array of counters, equivalence.

20030820

255

4.27 balance_modulo

Origin	Derived from <code>balance</code> .
Constraint	<code>balance_modulo(BALANCE, VARIABLES, M)</code>
Argument(s)	<pre> BALANCE : dvar VARIABLES : collection(var - dvar) M : int </pre>
Restriction(s)	<pre> BALANCE ≥ 0 BALANCE ≤ VARIABLES required(VARIABLES, var) M > 0 </pre>
Purpose	<p>Consider the largest set \mathcal{S}_1 (respectively the smallest set \mathcal{S}_2) of variables of the collection <code>VARIABLES</code> which have the same remainder when divided by <code>M</code>. <code>BALANCE</code> is equal to the difference between the cardinality of \mathcal{S}_2 and the cardinality of \mathcal{S}_1.</p>
Arc input(s)	<code>VARIABLES</code>
Arc generator	<code>CLIQUE</code> \mapsto <code>collection(variables1, variables2)</code>
Arc arity	2
Arc constraint(s)	<code>variables1.var mod M = variables2.var mod M</code>
Graph property(ies)	<code>RANGE_NSCC = BALANCE</code>
Example	$\text{balance_modulo} \left(2, \left\{ \begin{array}{l} \text{var} - 6, \\ \text{var} - 1, \\ \text{var} - 7, \\ \text{var} - 1, \\ \text{var} - 5 \end{array} \right\}, 3 \right)$ <p>In this example values 6, 1, 7, 1, 5 are respectively associated to the equivalence classes 0, 1, 1, 1, 2. Therefore the equivalence classes 0, 1 and 2 are respectively used 1, 3 and 1 times. <code>BALANCE</code> is assigned to the difference between the maximum and minimum number of the previous occurrences (i.e. 3 - 1). Parts (A) and (B) of Figure 4.56 respectively show the initial and final graph. Since we use the <code>RANGE_NSCC</code> graph property, we show the largest and smallest strongly connected components of the final graph.</p>
Graph model	The graph property <code>RANGE_NSCC</code> constraints the difference between the sizes of the largest and smallest strongly connected components.
Automaton	Figure 4.57 depicts the automaton associated to the <code>balance_modulo</code> constraint. To each item of the collection <code>VARIABLES</code> corresponds a signature variable S_i , which is equal to 1.

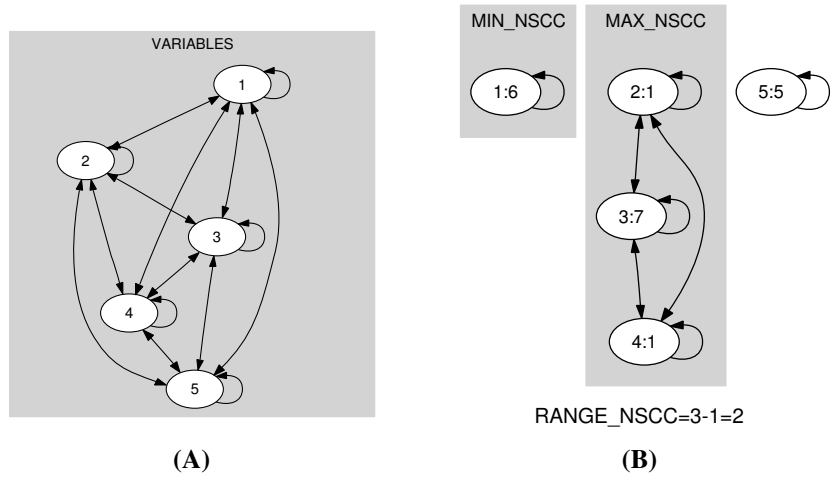


Figure 4.56: Initial and final graph of the balance_modulo constraint

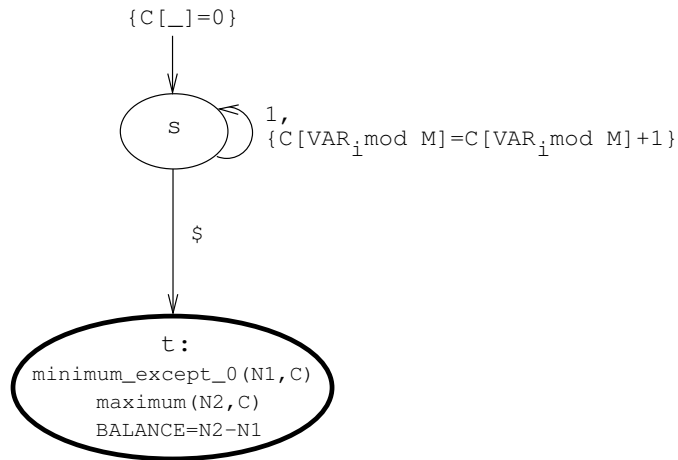


Figure 4.57: Automaton of the balance_modulo constraint

- Usage** One application of this constraint is to enforce a *balanced assignment* of values, no matter how many distinct equivalence classes will be used. In this case one will *push down* the maximum value of the first argument of the `balance_modulo` constraint.
- See also** `balance`.
- Key words** value constraint, modulo, assignment, balanced assignment, automaton, automaton with array of counters, equivalence.

20030820

259

4.28 balance_partition

Origin	Derived from balance.
Constraint	balance_partition(BALANCE, VARIABLES, PARTITIONS)
Type(s)	VALUES : collection(val - int)
Argument(s)	BALANCE : dvar VARIABLES : collection(var - dvar) PARTITIONS : collection(p - VALUES)
Restriction(s)	required(VALUES, val) distinct(VALUES, val) BALANCE ≥ 0 BALANCE ≤ VARIABLES required(VARIABLES, var) required(PARTITIONS, p) PARTITIONS ≥ 2
Purpose	Consider the largest set S_1 (respectively the smallest set S_2) of variables of the collection VARIABLES which take their value in the same partition of the collection PARTITIONS. BALANCE is equal to the difference between the cardinality of S_2 and the cardinality of S_1 .
Arc input(s)	VARIABLES
Arc generator	CLIQUE \mapsto collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	in_same_partition(variables1.var, variables2.var, PARTITIONS)
Graph property(ies)	RANGE_NSCC = BALANCE

Example	$\text{balance_partition} \left(1, \left(\begin{array}{l} \left\{ \begin{array}{l} \text{var} - 6, \\ \text{var} - 2, \\ \text{var} - 6, \\ \text{var} - 4, \\ \text{var} - 4 \end{array} \right\}, \\ \left\{ \begin{array}{l} \text{p} - \{\text{val} - 1, \text{val} - 3\}, \\ \text{p} - \{\text{val} - 4\}, \\ \text{p} - \{\text{val} - 2, \text{val} - 6\} \end{array} \right\} \end{array} \right) \right)$
----------------	--

In this example values 6, 2, 6, 4, 4 are respectively associated to the partitions $p - \{\text{val} - 2, \text{val} - 6\}$ and $p - \{\text{val} - 4\}$. Partitions $p - \{\text{val} - 4\}$ and $p - \{\text{val} - 2, \text{val} - 6\}$ are respectively used 2 and 3 times. BALANCE is assigned to the difference between the maximum and minimum number of the previous occurrences (i.e. 3 - 2). Note that we don't consider those partitions that are not used at all. Parts (A) and (B) of Figure 4.58 respectively show the initial and final graph. Since we use the RANGE_NSCC graph property, we show the largest and smallest strongly connected components of the final graph.

Graph model	The graph property RANGE_NSCC constraints the difference between the sizes of the largest and smallest strongly connected components.
Usage	One application of this constraint is to enforce a <i>balanced assignment</i> of values, no matter how many distinct partitions will be used. In this case one will <i>push down</i> the maximum value of the first argument of the <code>balance_partition</code> constraint.
See also	<code>balance</code> .
Key words	value constraint, partition, assignment, balanced assignment, equivalence.

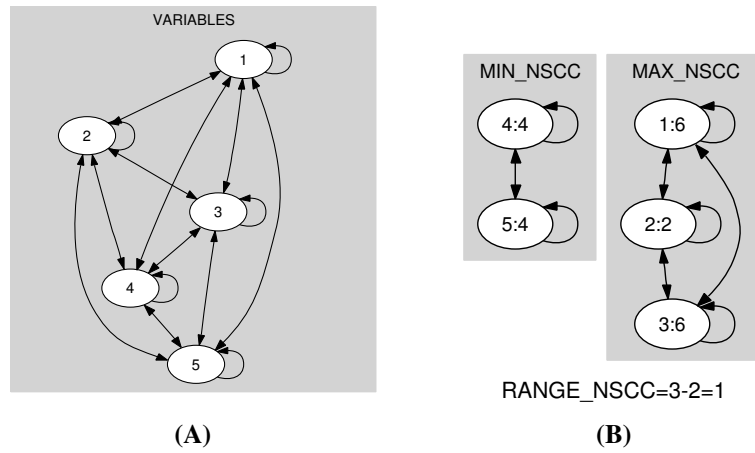


Figure 4.58: Initial and final graph of the balance_partition constraint

4.29 bin_packing

Origin	Derived from cumulative.
Constraint	<code>bin_packing(CAPACITY, ITEMS)</code>
Argument(s)	<code>CAPACITY</code> : int <code>ITEMS</code> : collection(bin – dvar, weight – int)
Restriction(s)	<code>CAPACITY</code> ≥ 0 <code>required(ITEMS, [bin, weight])</code> <code>ITEMS.weight</code> ≥ 0 <code>ITEMS.weight</code> \leq <code>CAPACITY</code>
Purpose	Given several items of the collection <code>ITEMS</code> (each of them having a specific weight), and different bins of a fixed capacity, assign each item to a bin so that the total weight of the items in each bin does not exceed <code>CAPACITY</code> .
Arc input(s)	<code>ITEMS ITEMS</code>
Arc generator	<code>PRODUCT</code> \mapsto <code>collection(items1, items2)</code>
Arc arity	2
Arc constraint(s)	<code>items1.bin = items2.bin</code>
Sets	$\text{SUCC} \mapsto \left[\begin{array}{l} \text{source,} \\ \text{variables} - \text{col} \left(\begin{array}{l} \text{VARIABLES} - \text{collection}(\text{var} - \text{dvar}), \\ [\text{item}(\text{var} - \text{ITEMS.weight})] \end{array} \right) \end{array} \right]$
Constraint(s) on sets	<code>sum_ctr(variables, \leq, CAPACITY)</code>
Example	$\text{bin_packing} \left(5, \left\{ \begin{array}{ll} \text{bin} - 3 & \text{weight} - 4, \\ \text{bin} - 1 & \text{weight} - 3, \\ \text{bin} - 3 & \text{weight} - 1 \end{array} \right\} \right)$ <p>Parts (A) and (B) of Figure 4.59 respectively show the initial and final graph. Each connected component of the final graph corresponds to the items which are all assigned to the same bin. The <code>bin_packing</code> constraint holds since the sum of the height of items which are assigned to bins 1 and 3 is respectively equal to 3 and 5. The previous quantities are both less than or equal to the maximum <code>CAPACITY</code> 5. Figure 4.60 shows the solution associated to the previous example.</p>
Graph model	We enforce the <code>sum_ctr</code> constraint on the weight of the items that are assigned to the same bin.
Automaton	Figure 4.61 depicts the automaton associated to the <code>bin_packing</code> constraint. To each item of the collection <code>ITEMS</code> corresponds a signature variable S_i , which is equal to 1.

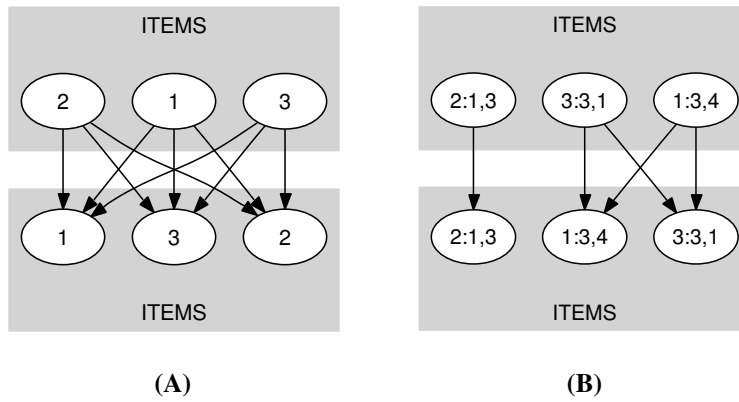


Figure 4.59: Initial and final graph of the bin_packing constraint

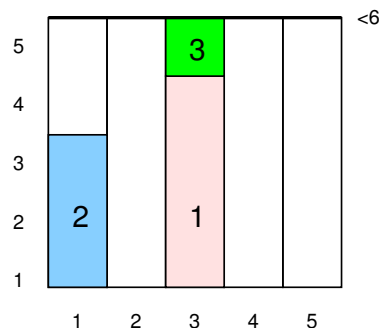


Figure 4.60: Bin-packing solution

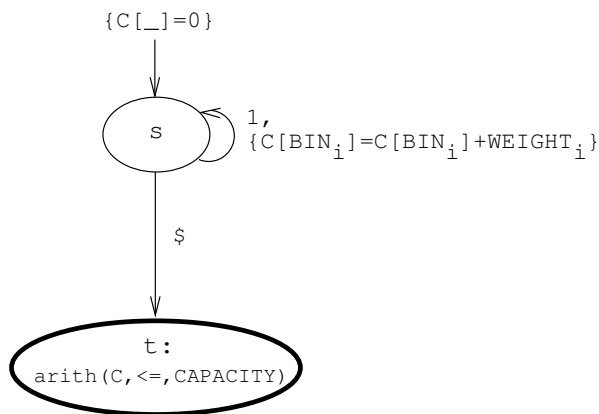


Figure 4.61: Automaton of the bin_packing constraint

- Remark** Note the difference with the *classical* bin-packing problem [66, page 221] where one wants to find solutions that minimize the number of bins. In our case each item may be assigned only to specific bins (i.e. the different values of the bin variable) and the goal is to find a feasible solution. This constraint can be seen as a special case of the `cumulative` constraint [67], where all tasks durations are equal to one.
- In [68] the `CAPACITY` parameter of the `bin_packing` constraint is replaced by a collection of domain variables representing the *load* of each bin (i.e. the sum of the weights of the items assigned to a bin). This allows representing problems where a minimum level has to be reached in each bin.
- Algorithm** [69, 70, 71, 72, 68].
- See also** `cumulative`.
- Key words** resource constraint, assignment, automaton, automaton with array of counters.

20000128

267

4.30 binary_tree

Origin	Derived from tree.
Constraint	binary_tree(NTREES, NODES)
Argument(s)	NTREES : dvar NODES : collection(index - int, succ - dvar)
Restriction(s)	NTREES \geq 0 required(NODES, [index, succ]) NODES.index \geq 1 NODES.index \leq NODES distinct(NODES, index) NODES.succ \geq 1 NODES.succ \leq NODES
Purpose	Cover the digraph G described by the NODES collection with NTREES binary trees in such a way that each vertex of G belongs to one distinct binary tree. The edges of the binary trees are directed from their leaves to their respective root.
Arc input(s)	NODES
Arc generator	$CLIQUE \mapsto$ collection(nodes1, nodes2)
Arc arity	2
Arc constraint(s)	nodes1.succ = nodes2.index
Graph property(ies)	<ul style="list-style-type: none"> • MAX_NSCC \leq 1 • NCC = NTREES • MAX_ID \leq 2

Example	$\text{binary_tree} \left(2, \left\{ \begin{array}{l} \text{index} - 1 \quad \text{succ} - 1, \\ \text{index} - 2 \quad \text{succ} - 3, \\ \text{index} - 3 \quad \text{succ} - 5, \\ \text{index} - 4 \quad \text{succ} - 7, \\ \text{index} - 5 \quad \text{succ} - 1, \\ \text{index} - 6 \quad \text{succ} - 1, \\ \text{index} - 7 \quad \text{succ} - 7, \\ \text{index} - 8 \quad \text{succ} - 5 \end{array} \right\} \right)$
----------------	---

Parts (A) and (B) of Figure 4.62 respectively show the initial and final graph. Since we use the **NCC** graph property, we display the two connected components of the final graph. Each of them corresponds to a binary tree. Since we use the **MAX_ID** graph property, we also show with a double circle a vertex which has a maximum number of predecessors.

The **binary_tree** constraint holds since all strongly connected components of the final graph have no more than one vertex, since $\text{NTREES} = \text{NCC} = 2$ and since $\text{MAX_ID} = 2$.

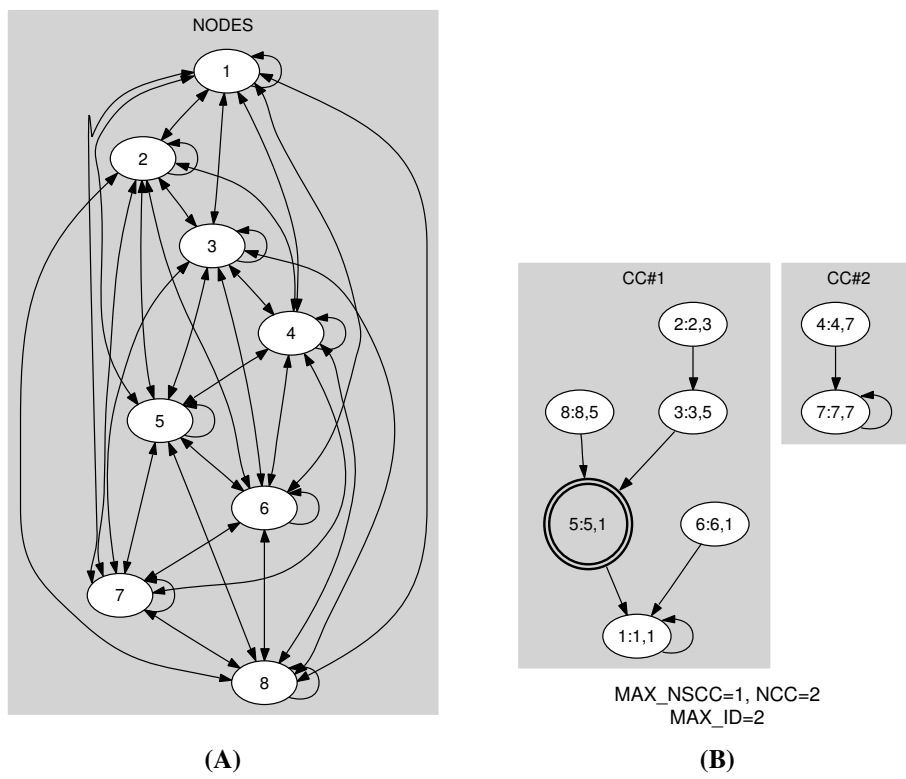


Figure 4.62: Initial and final graph of the binary_tree constraint

- Graph model** We use the same graph constraint as for the `tree` constraint, except that we add the graph property $\text{MAX_ID} \leq 2$ which constraints the maximum in-degree of the final graph to not exceed 2. `MAX_ID` does not consider loops: This is why we do not have any problem with the root of each tree.
- See also** `tree`.
- Key words** graph constraint, graph partitioning constraint, connected component, tree, `one_succ`.

20000128

271

4.31 cardinality_atleast

Origin	Derived from <code>global_cardinality</code> .
Constraint	<code>cardinality_atleast(ATLEAST, VARIABLES, VALUES)</code>
Argument(s)	<code>ATLEAST</code> : dvar <code>VARIABLES</code> : <code>collection(var - dvar)</code> <code>VALUES</code> : <code>collection(val - int)</code>
Restriction(s)	<code>ATLEAST</code> ≥ 0 <code>ATLEAST</code> $\leq VARIABLES $ <code>required(VARIABLES, var)</code> <code>required(VALUES, val)</code> <code>distinct(VALUES, val)</code>
Purpose	<code>ATLEAST</code> is the minimum number of time that a value of <code>VALUES</code> is taken by the variables of the collection <code>VARIABLES</code> .
Arc input(s)	<code>VARIABLES VALUES</code>
Arc generator	<code>PRODUCT</code> \mapsto <code>collection(variables, values)</code>
Arc arity	2
Arc constraint(s)	<code>variables.var</code> \neq <code>values.val</code>
Graph property(ies)	<code>MAX_ID</code> = $ VARIABLES - ATLEAST$
Example	$\text{cardinality_atleast} \left(1, \{ \text{var} - 3, \text{var} - 3, \text{var} - 8 \}, \{ \text{val} - 3, \text{val} - 8 \} \right)$ <p>In this example, values 3 and 8 are respectively used 2, and 1 times. Therefore <code>ATLEAST</code> is assigned to $3 - 2 = 1$. Parts (A) and (B) of Figure 4.63 respectively show the initial and final graph. Since we use the <code>MAX_ID</code> graph property, the vertex with the maximum number of predecessor is stressed with a double circle.</p>
Graph model	Using directly the graph property <code>MIN_ID</code> = <code>ATLEAST</code> and replacing the disequality of the arc constraint by an equality does not work since it ignores values which are not assigned to any variable. This comes from the fact that isolated vertices are removed from the final graph.
Automaton	Figure 4.64 depicts the automaton associated to the <code>cardinality_atleast</code> constraint. To each variable <code>VAR_i</code> of the collection <code>VARIABLES</code> corresponds a 0-1 signature variable <code>S_i</code> . The following signature constraint links <code>VAR_i</code> and <code>S_i</code> : <code>VAR_i ∈ VALUES</code> \Leftrightarrow <code>S_i</code> .
Usage	An application of this constraint is to enforce a minimum use of values.

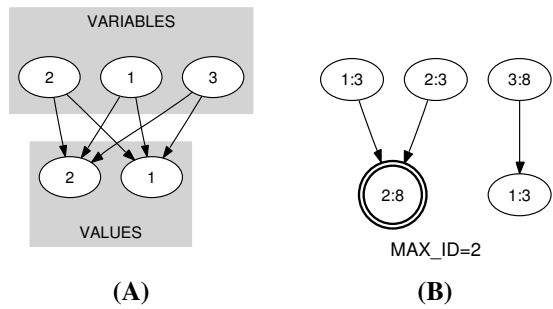


Figure 4.63: Initial and final graph of the cardinality_atleast constraint

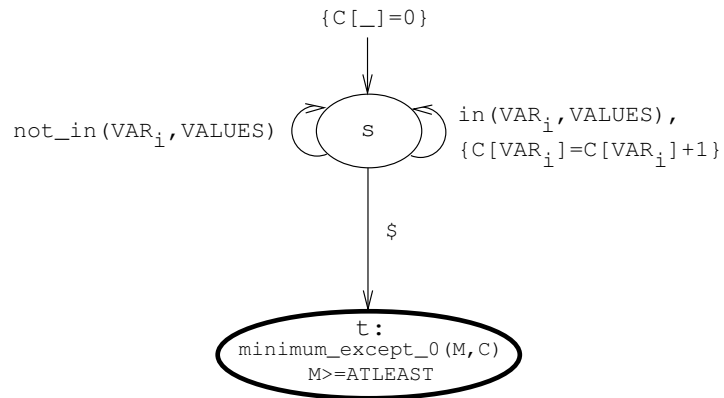


Figure 4.64: Automaton of the cardinality_atleast constraint

- Remark** This is a restricted form of a variant of an `among` constraint and of the `global_cardinality` constraint. In the original `global_cardinality` constraint, one specifies for each value its minimum and maximum number of occurrences.
- Algorithm** See `global_cardinality` [19].
- See also** `global_cardinality`.
- Key words** value constraint, assignment, at least, automaton, automaton with array of counters, acyclic, bipartite, no_loop.

20030820

275

4.32 cardinality_atmost

Origin	Derived from <code>global_cardinality</code> .
Constraint	<code>cardinality_atmost(ATMOST, VARIABLES, VALUES)</code>
Argument(s)	<code>ATMOST</code> : dvar <code>VARIABLES</code> : collection(var – dvar) <code>VALUES</code> : collection(val – int)
Restriction(s)	<code>ATMOST</code> ≥ 0 <code>ATMOST</code> $\leq VARIABLES $ <code>required(VARIABLES, var)</code> <code>required(VALUES, val)</code> <code>distinct(VALUES, val)</code>
Purpose	<code>ATMOST</code> is the maximum number of occurrences of each value of <code>VALUES</code> within the variables of the collection <code>VARIABLES</code> .
Arc input(s)	<code>VARIABLES VALUES</code>
Arc generator	<code>PRODUCT</code> \mapsto <code>collection(variables, values)</code>
Arc arity	2
Arc constraint(s)	<code>variables.var = values.val</code>
Graph property(ies)	<code>MAX_ID = ATMOST</code>

Example	<code>cardinality_atmost</code> $\left(2, \left\{ \begin{array}{l} \text{var} - 2, \\ \text{var} - 1, \\ \text{var} - 7, \\ \text{var} - 1, \\ \text{var} - 2 \end{array} \right\}, \left\{ \begin{array}{l} \text{val} - 5, \\ \text{val} - 7, \\ \text{val} - 2, \\ \text{val} - 9 \end{array} \right\} \right)$
----------------	---

In this example, values 5, 7, 2 and 9 are respectively used 0, 1, 2 and 0 times. Therefore `ATMOST` is assigned to the maximum number of occurrences 2. Parts (A) and (B) of Figure 4.65 respectively show the initial and final graph. Since we use the `MAX_ID` graph property, the vertex which has the maximum number of predecessor is stressed with a double circle.

Automaton Figure 4.66 depicts the automaton associated to the `cardinality_atmost` constraint. To each variable `VARi` of the collection `VARIABLES` corresponds a 0-1 signature variable `Si`. The following signature constraint links `VARi` and `Si`: `VARi ∈ VALUES ⇔ Si`.

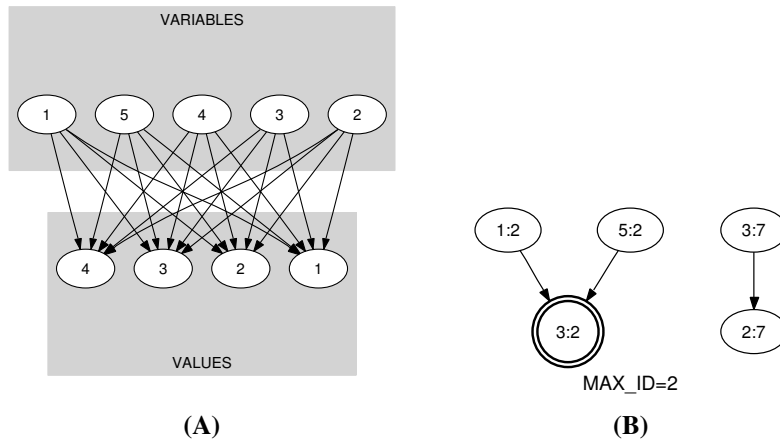


Figure 4.65: Initial and final graph of the cardinality_atmost constraint

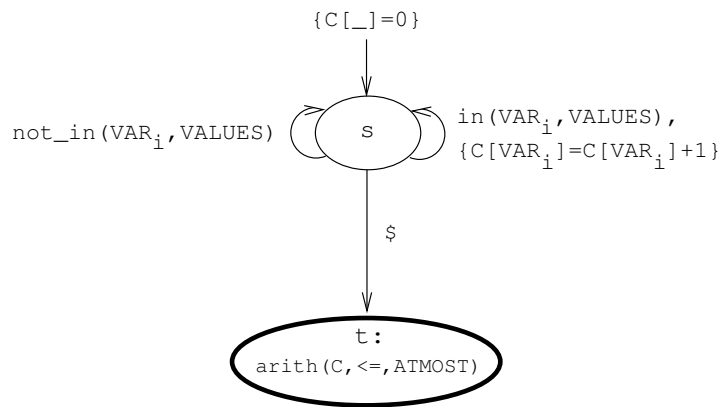


Figure 4.66: Automaton of the cardinality_atmost constraint

Usage	One application of this constraint is to enforce a maximum use of values.
Remark	This is a restricted form of a variant of the <code>among</code> constraint and of the <code>global_cardinality</code> constraint. In the original <code>global_cardinality</code> constraint, one specifies for each value its minimum and maximum number of occurrences.
Algorithm	See <code>global_cardinality</code> [19].
See also	<code>global_cardinality</code> .
Key words	value constraint, assignment, at most, automaton, automaton with array of counters, acyclic, bipartite, no_loop.

20030820

279

4.33 cardinality_atmost_partition

Origin	Derived from <code>global_cardinality</code> .
Constraint	<code>cardinality_atmost_partition(ATMOST, VARIABLES, PARTITIONS)</code>
Type(s)	<code>VALUES : collection(val - int)</code>
Argument(s)	<code>ATMOST : dvar</code> <code>VARIABLES : collection(var - dvar)</code> <code>PARTITIONS : collection(p - VALUES)</code>
Restriction(s)	<code>required(VALUES, val)</code> <code>distinct(VALUES, val)</code> <code>ATMOST ≥ 0</code> <code>ATMOST ≤ VARIABLES </code> <code>required(VARIABLES, var)</code> <code>required(PARTITIONS, p)</code> <code> PARTITIONS ≥ 2</code>
Purpose	ATMOST is the maximum number of time that values of a same partition of PARTITIONS are taken by the variables of the collection VARIABLES.
Arc input(s)	VARIABLES PARTITIONS
Arc generator	<i>PRODUCT</i> \mapsto <code>collection(variables, partitions)</code>
Arc arity	2
Arc constraint(s)	<code>in(variables.var, partitions.p)</code>
Graph property(ies)	<code>MAX_ID = ATMOST</code>

Example	<code>cardinality_atmost_partition</code> $\left(2, \left\{ \begin{array}{l} \text{var} - 2, \\ \text{var} - 3, \\ \text{var} - 7, \\ \text{var} - 1, \\ \text{var} - 6, \\ \text{var} - 0 \end{array} \right\}, \left\{ \begin{array}{l} p - \{\text{val} - 1, \text{val} - 3\}, \\ p - \{\text{val} - 4\}, \\ p - \{\text{val} - 2, \text{val} - 6\} \end{array} \right\} \right)$
----------------	---

In this example, two variables are assigned to values of the first partition, no variable is assigned to a value of the second partition, and finally two variables are assigned to values of the last partition. Therefore ATMOST is assigned to the maximum number of occurrences 2. Parts (A) and (B) of Figure 4.67 respectively show the initial and final graph. Since we use the `MAX_ID` graph property, a vertex with the maximum number of predecessor is stressed with a double circle.

20030820

281

See also `global_cardinality`, in.

Key words value constraint, partition, at most, acyclic, bipartite, no_loop.

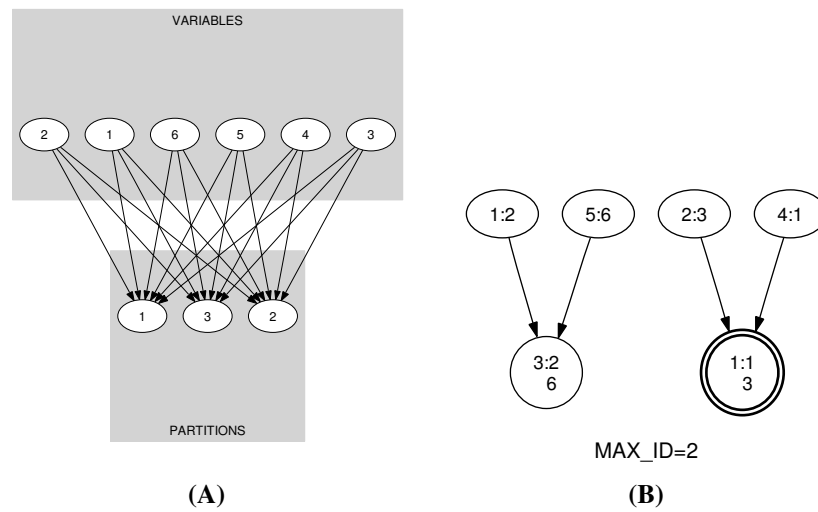


Figure 4.67: Initial and final graph of the `cardinality_atmost_partition` constraint

4.34 change

Origin	CHIP
Constraint	$\text{change}(\text{NCHANGE}, \text{VARIABLES}, \text{CTR})$
Synonym(s)	nbchanges , similarity .
Argument(s)	$\text{NCHANGE} \quad : \text{dvar}$ $\text{VARIABLES} \quad : \text{collection}(\text{var} - \text{dvar})$ $\text{CTR} \quad : \text{atom}$
Restriction(s)	$\text{NCHANGE} \geq 0$ $\text{NCHANGE} < \text{VARIABLES} $ $\text{required}(\text{VARIABLES}, \text{var})$ $\text{CTR} \in [=, \neq, <, \geq, >, \leq]$
Purpose	NCHANGE is the number of times that constraint CTR holds on consecutive variables of the collection VARIABLES .
Arc input(s)	VARIABLES
Arc generator	$\text{PATH} \mapsto \text{collection}(\text{variables1}, \text{variables2})$
Arc arity	2
Arc constraint(s)	$\text{variables1.var CTR variables2.var}$
Graph property(ies)	$\text{NARC} = \text{NCHANGE}$

Example	$\text{change} \left(3, \left\{ \begin{array}{l} \text{var} - 4, \\ \text{var} - 4, \\ \text{var} - 3, \\ \text{var} - 4, \\ \text{var} - 1 \end{array} \right\}, \neq \right)$ $\text{change} \left(1, \left\{ \begin{array}{l} \text{var} - 1, \\ \text{var} - 2, \\ \text{var} - 4, \\ \text{var} - 3, \\ \text{var} - 7 \end{array} \right\}, > \right)$
----------------	--

In the first example the changes are located between values 4 and 3, 3 and 4, 4 and 1. In the second example the unique change occurs between values 4 and 3. Parts (A) and (B) of Figure 4.68 respectively show the initial and final graph of the first example. Since we use the **NARC** graph property, the arcs of the final graph are stressed in bold.

Graph model	Since we are only interested by the constraints linking two consecutive items of the collection VARIABLES we use PATH to generate the arcs of the initial graph.
--------------------	--

Automaton	Figure 4.69 depicts the automaton associated to the <code>change</code> constraint. To each pair of consecutive variables (VAR_i, VAR_{i+1}) of the collection <code>VARIABLES</code> corresponds a 0-1 signature variable S_i . The following signature constraint links VAR_i , VAR_{i+1} and S_i : $VAR_i \text{ CTR } VAR_{i+1} \Leftrightarrow S_i$.
Usage	This constraint can be used in the context of timetabling problems in order to put an upper limit on the number of changes of job types during a given period.
Remark	A similar constraint appears in [73, page 338] under the name of <code>similarity</code> constraint. The difference consists of replacing the arithmetic constraint <code>CTR</code> by a binary constraint. When <code>CTR</code> is equal to \neq this constraint is called <code>nbchanges</code> in [40].
Algorithm	[65].
Used in	<code>pattern</code> .
See also	<code>smooth</code> , <code>change_partition</code> , <code>change_pair</code> , <code>circular_change</code> , <code>longest_change</code> .
Key words	timetabling constraint, number of changes, automaton, automaton with counters, sliding cyclic(1) constraint network(2), sliding cyclic(1) constraint network(3), acyclic, no_loop.

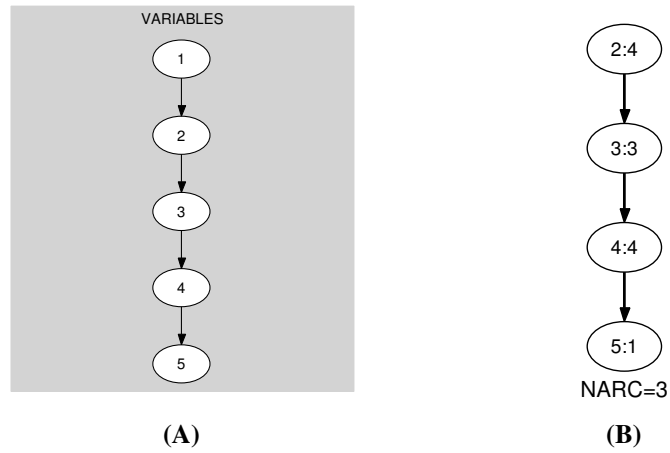


Figure 4.68: Initial and final graph of the change constraint

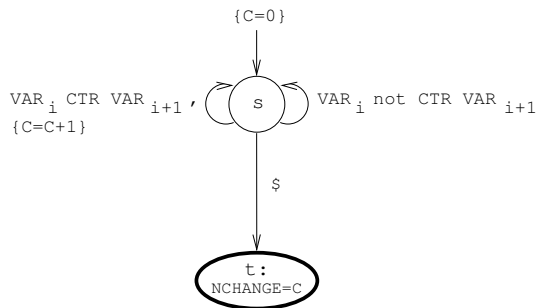


Figure 4.69: Automaton of the change constraint

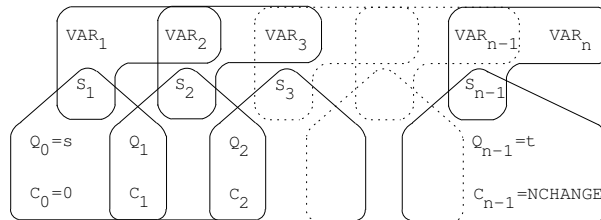


Figure 4.70: Hypergraph of the reformulation corresponding to the automaton of the change constraint

20000128

287

4.35 change_continuity

Origin N. Beldiceanu

Constraint	change_continuity	$\left(\begin{array}{l} \text{NB_PERIOD_CHANGE,} \\ \text{NB_PERIOD_CONTINUITY,} \\ \text{MIN_SIZE_CHANGE,} \\ \text{MAX_SIZE_CHANGE,} \\ \text{MIN_SIZE_CONTINUITY,} \\ \text{MAX_SIZE_CONTINUITY,} \\ \text{NB_CHANGE,} \\ \text{NB_CONTINUITY,} \\ \text{VARIABLES,} \\ \text{CTR} \end{array} \right)$
-------------------	-------------------	---

Argument(s)	NB_PERIOD_CHANGE : dvar NB_PERIOD_CONTINUITY : dvar MIN_SIZE_CHANGE : dvar MAX_SIZE_CHANGE : dvar MIN_SIZE_CONTINUITY : dvar MAX_SIZE_CONTINUITY : dvar NB_CHANGE : dvar NB_CONTINUITY : dvar VARIABLES : collection(var – dvar) CTR : atom
--------------------	--

Restriction(s)	NB_PERIOD_CHANGE \geq 0 NB_PERIOD_CONTINUITY \geq 0 MIN_SIZE_CHANGE \geq 0 MAX_SIZE_CHANGE \geq MIN_SIZE_CHANGE MIN_SIZE_CONTINUITY \geq 0 MAX_SIZE_CONTINUITY \geq MIN_SIZE_CONTINUITY NB_CHANGE \geq 0 NB_CONTINUITY \geq 0 required(VARIABLES, var) CTR \in [=, \neq , <, \geq , >, \leq]
-----------------------	--

On the one hand a *change* is defined by the fact that constraint $\text{VARIABLES}[i].\text{var CTR VARIABLE}[i + 1].\text{var}$ holds.
 On the other hand a *continuity* is defined by the fact that constraint $\text{VARIABLES}[i].\text{var CTR VARIABLE}[i + 1].\text{var}$ does not hold.
 A *period of change* on variables

$$\text{VARIABLES}[i].\text{var}, \text{VARIABLES}[i + 1].\text{var}, \dots, \text{VARIABLES}[j].\text{var} \quad (i < j)$$

is defined by the fact that all constraints $\text{VARIABLES}[k].\text{var CTR VARIABLE}[k + 1].\text{var}$ hold for $k \in [i, j - 1]$.

A *period of continuity* on variables

$$\text{VARIABLES}[i].\text{var}, \text{VARIABLES}[i + 1].\text{var}, \dots, \text{VARIABLES}[j].\text{var} \quad (i < j)$$

is defined by the fact that all constraints $\text{VARIABLES}[k].\text{var CTR VARIABLE}[k + 1].\text{var}$ do not hold for $k \in [i, j - 1]$.

The constraint *change.continuity* holds if and only if:

- **NB_PERIOD_CHANGE** is equal to the number of periods of change,
- **NB_PERIOD_CONTINUITY** is equal to the number of periods of continuity,
- **MIN_SIZE_CHANGE** is equal to the number of variables of the smallest period of change,
- **MAX_SIZE_CHANGE** is equal to the number of variables of the largest period of change,
- **MIN_SIZE_CONTINUITY** is equal to the number of variables of the smallest period of continuity,
- **MAX_SIZE_CONTINUITY** is equal to the number of variables of the largest period of continuity,
- **NB_CHANGE** is equal to the total number of changes,
- **NB_CONTINUITY** is equal to the total number of continuities.

Purpose

Arc input(s)	VARIABLES
Arc generator	$PATH \mapsto \text{collection}(\text{variables1}, \text{variables2})$
Arc arity	2
Arc constraint(s)	$\text{variables1}.\text{var CTR variables2}.\text{var}$
Graph property(ies)	<ul style="list-style-type: none"> • NCC = NB_PERIOD_CHANGE • MIN_NCC = MIN_SIZE_CHANGE • MAX_NCC = MAX_SIZE_CHANGE • NARC = NB_CHANGE

Arc input(s)	VARIABLES
Arc generator	$PATH \mapsto \text{collection}(\text{variables1}, \text{variables2})$
Arc arity	2
Arc constraint(s)	$\text{variables1}.\text{var} \neg \text{CTR variables2}.\text{var}$

- Graph property(ies)**
- `NCC = NB_PERIOD_CONTINUITY`
 - `MIN_NCC = MIN_SIZE_CONTINUITY`
 - `MAX_NCC = MAX_SIZE_CONTINUITY`
 - `NARC = NB_CONTINUITY`

Example

$$\text{change_continuity} \left(\begin{array}{c} 3, 2, 2, 4, 2, 4, 6, 4, \\ \left. \begin{array}{l} \text{var} - 1, \\ \text{var} - 3, \\ \text{var} - 1, \\ \text{var} - 8, \\ \text{var} - 8, \\ \text{var} - 4, \\ \text{var} - 7, \\ \text{var} - 7, \\ \text{var} - 7, \\ \text{var} - 7, \\ \text{var} - 2 \end{array} \right\} \right) \neq$$

Figure 4.71 makes clear the different parameters that are associated to the given example. We place character | for representing a change and a blank for a continuity. On top of the solution we represent the different periods of change, while below we show the different periods of continuity. Parts (A) and (B) of Figure 4.72 respectively show the initial and final graph associated to the first graph constraint.

```

<-----> <---->      <->
1|3|1|8 8|4|7 7 7 7|2
      <->      <----->

```

Figure 4.71: Periods of changes and periods of continuities

- Graph model** We use two graph constraints to respectively catch the constraints on the period of changes and of the period of continuities. In both case each period corresponds to a connected component of the final graph.
- Automaton** Figures 4.73 , 4.74 , 4.77 , 4.78 , 4.81 , 4.82 and 4.85 depict the automata associated to the different graph characteristics of the `change_continuity` constraint. For the automata that respectively compute `NB_PERIOD_CHANGE`, `NB_PERIOD_CONTINUITY` `MIN_SIZE_CHANGE`, `MIN_SIZE_CONTINUITY` `MAX_SIZE_CHANGE`, `MAX_SIZE_CONTINUITY` `NB_CHANGE` and `NB_CONTINUITY` we have a 0-1 signature variable S_i for each pair of consecutive variables (VAR_i, VAR_{i+1}) of the collection `VARIABLES`. The following signature constraint links VAR_i , VAR_{i+1} and S_i : $VAR_i \text{ CTR } VAR_{i+1} \Leftrightarrow S_i$.
- Remark** If the variables of the collection `VARIABLES` have to take distinct values between 1 and the total number of variables, we have what is called a permutation. In this case, if we choose the binary constraint `<`, then `MAX_SIZE_CHANGE` gives the size of the longest run of the permutation; A *run* is a maximal increasing contiguous subsequence in a permutation.
- See also** `group`, `group_skip_isolated_item`, `stretch_path`.
- Key words** timetabling constraint, run of a permutation, permutation, connected component, automaton, automaton with counters, sliding cyclic(1) constraint network(2), sliding cyclic(1) constraint network(3), acyclic, no_loop, apartition.

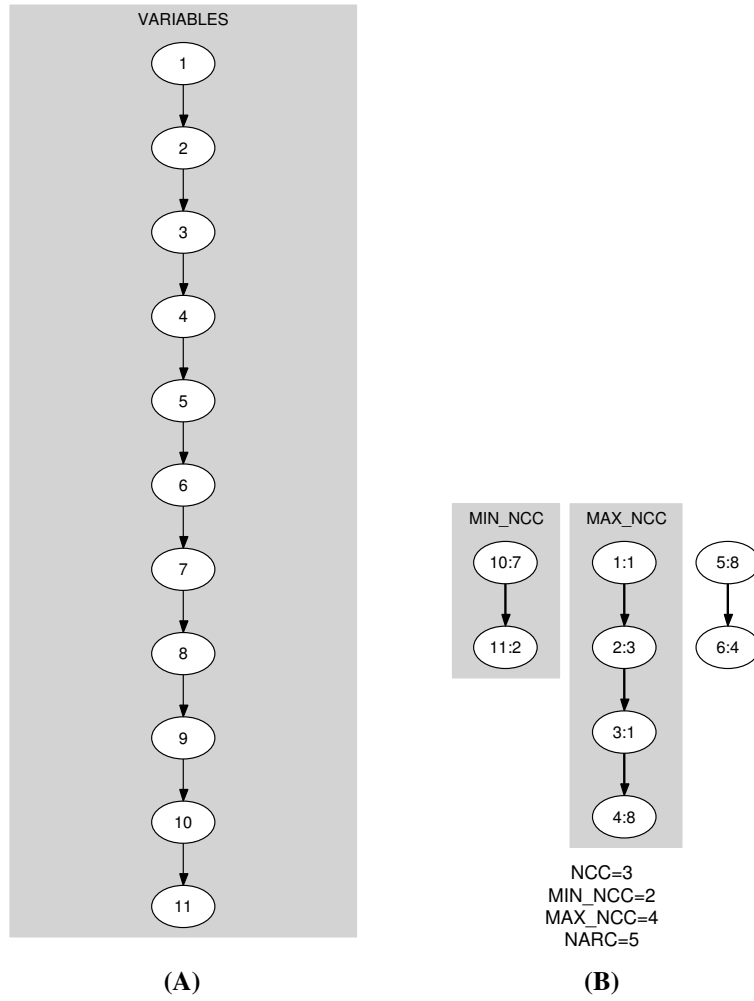


Figure 4.72: Initial and final graph of the change_continuity constraint

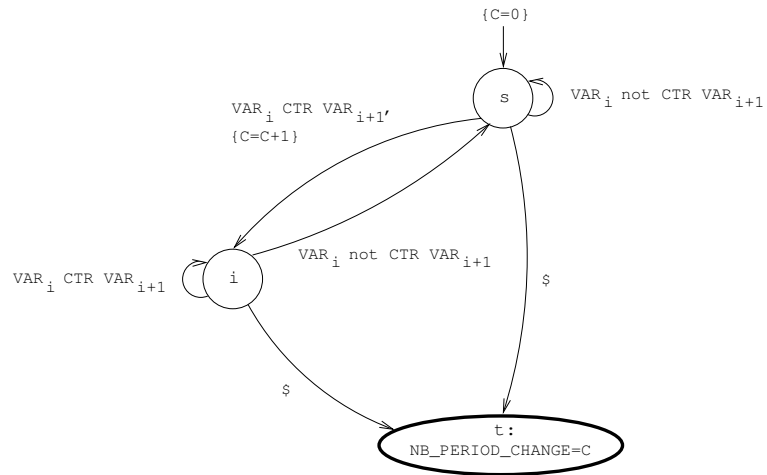


Figure 4.73: Automaton for the NB_PERIOD_CHANGE parameter of the change_continuity constraint

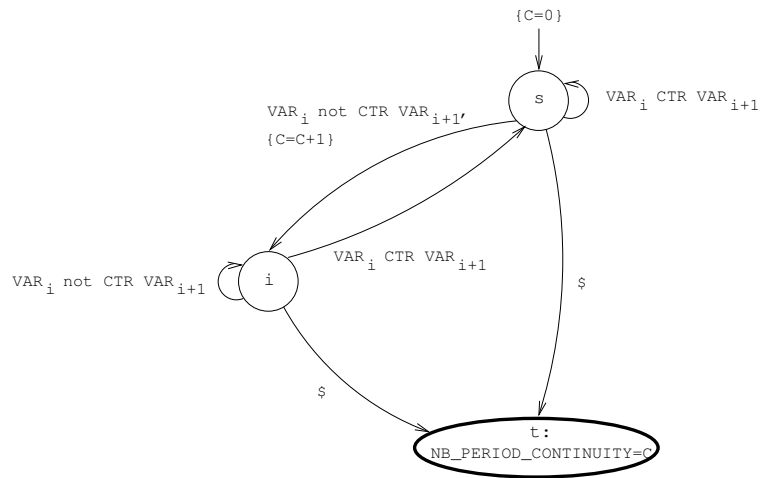


Figure 4.74: Automaton for the NB_PERIOD_CONTINUITY parameter of the change_continuity constraint

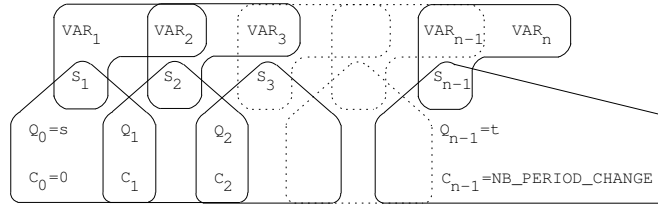


Figure 4.75: Hypergraph of the reformulation corresponding to the automaton of the NB_PERIOD_CHANGE parameter of the change_continuity constraint

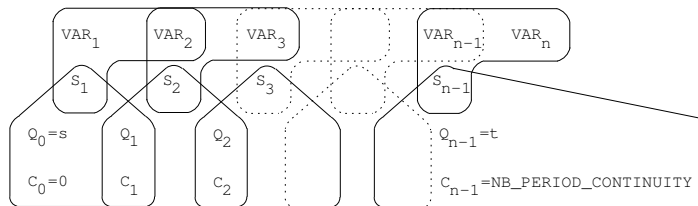


Figure 4.76: Hypergraph of the reformulation corresponding to the automaton of the NB_PERIOD_CONTINUITY parameter of the change_continuity constraint

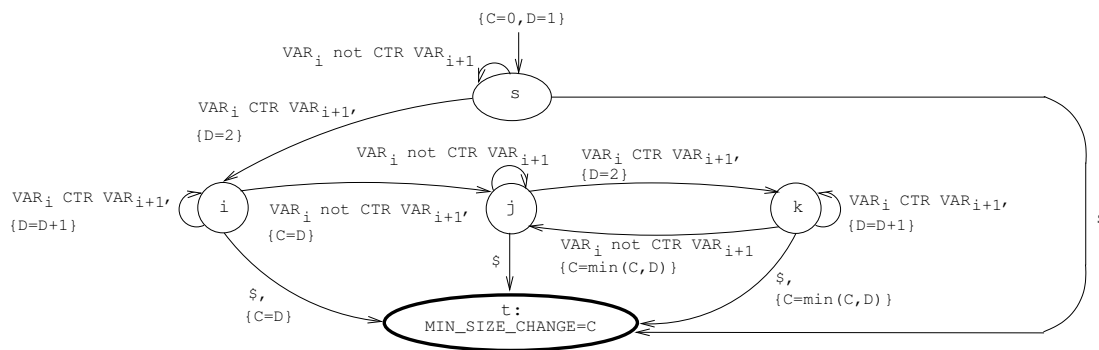


Figure 4.77: Automaton for the MIN_SIZE_CHANGE parameter of the change_continuity constraint

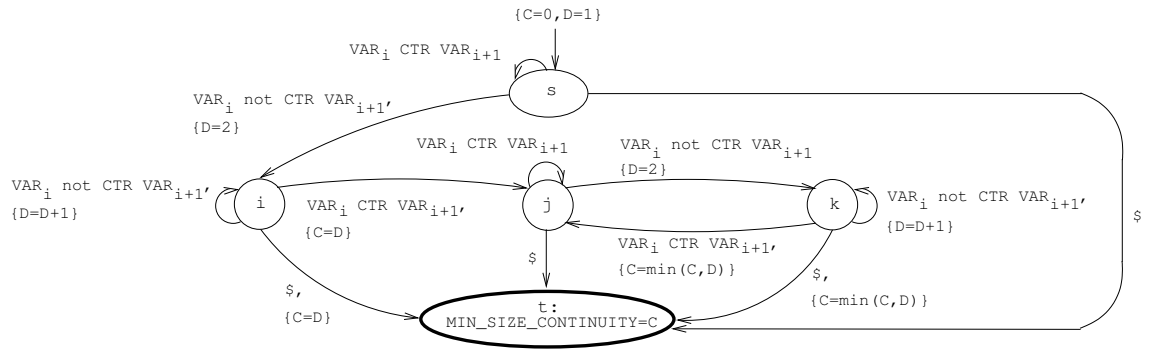


Figure 4.78: Automaton for the MIN_SIZE_CONTINUITY parameter of the change_continuity constraint

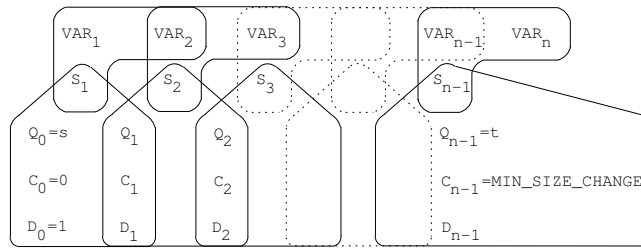


Figure 4.79: Hypergraph of the reformulation corresponding to the automaton of the MIN_SIZE_CHANGE parameter of the change_continuity constraint

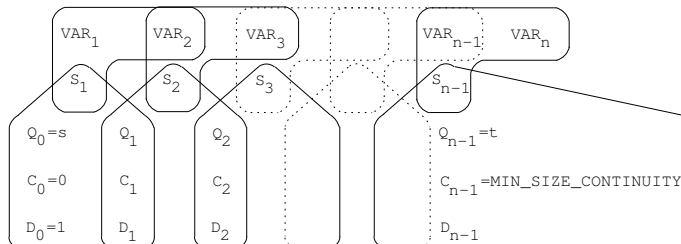


Figure 4.80: Hypergraph of the reformulation corresponding to the automaton of the MIN_SIZE_CONTINUITY parameter of the change_continuity constraint

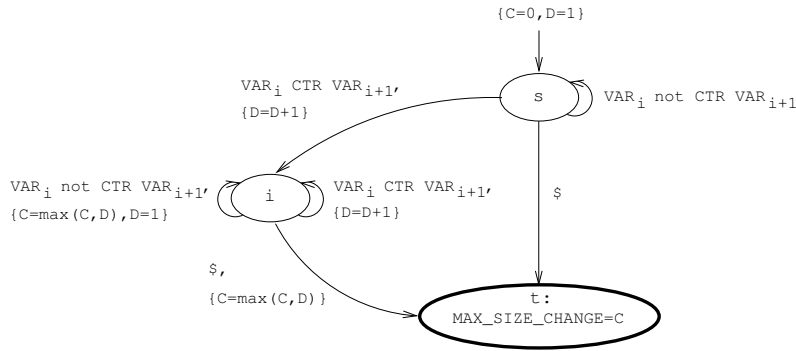


Figure 4.81: Automaton for the MAX_SIZE_CHANGE parameter of the change_continuity constraint

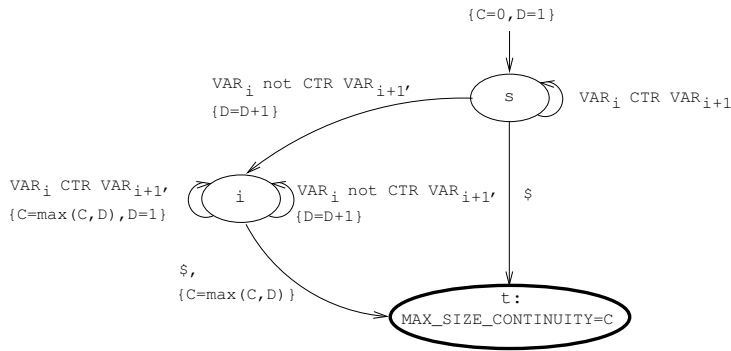


Figure 4.82: Automaton for the MAX_SIZE_CONTINUITY parameter of the change_continuity constraint

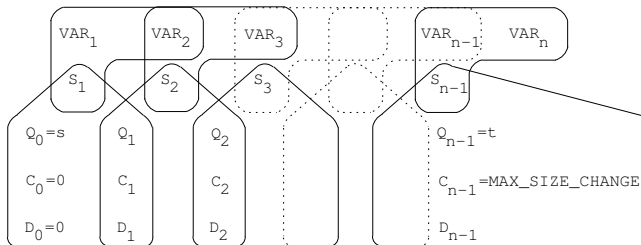


Figure 4.83: Hypergraph of the reformulation corresponding to the automaton of the MAX_SIZE_CHANGE parameter of the change_continuity constraint

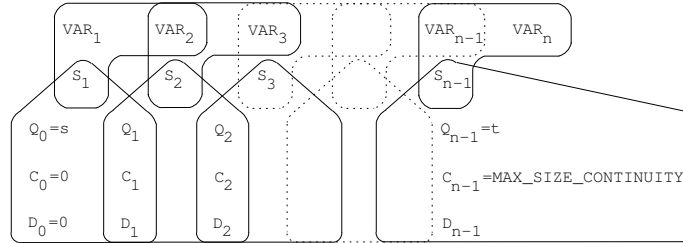


Figure 4.84: Hypergraph of the reformulation corresponding to the automaton of the `MAX_SIZE_CONTINUITY` parameter of the `change_continuity` constraint

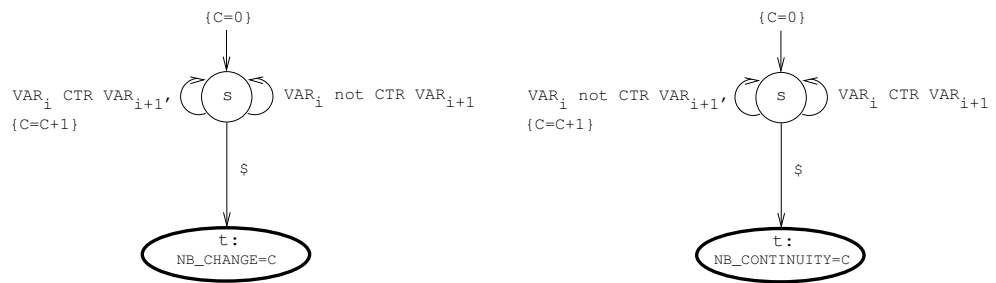


Figure 4.85: Automata for the `NB_CHANGE` and `NB_CONTINUITY` parameters of the `change_continuity` constraint

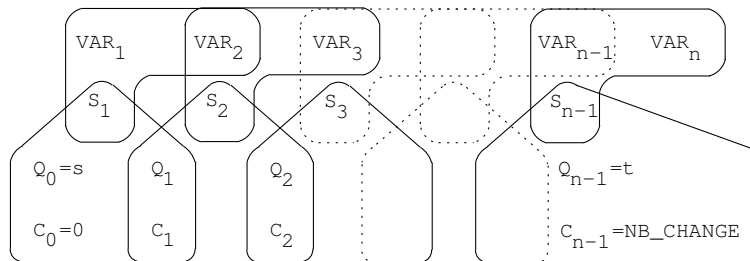


Figure 4.86: Hypergraph of the reformulation corresponding to the automaton of the `NB_CHANGE` parameter of the `change_continuity` constraint

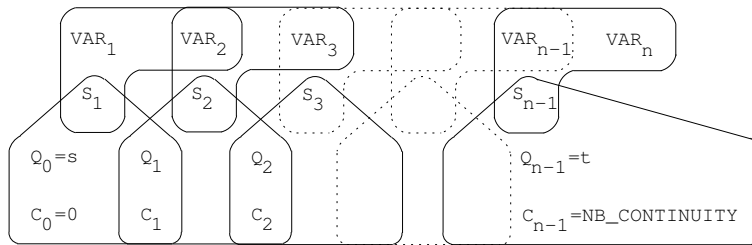


Figure 4.87: Hypergraph of the reformulation corresponding to the automaton of the NB_CONTINUITY parameter of the change_continuity constraint

4.36 change_pair

Origin	Derived from change.
Constraint	<code>change_pair(NCHANGE, PAIRS, CTRX, CTRY)</code>
Argument(s)	NCHANGE : dvar PAIRS : collection(x - dvar, y - dvar) CTRX : atom CTRY : atom
Restriction(s)	NCHANGE ≥ 0 NCHANGE $< PAIRS $ required(PAIRS, [x, y]) CTRX $\in [=, \neq, <, \geq, >, \leq]$ CTRY $\in [=, \neq, <, \geq, >, \leq]$
Purpose	NCHANGE is the number of times that the following disjunction holds: $(X_1 \text{ CTRX } X_2) \vee (Y_1 \text{ CTRY } Y_2)$, where (X_1, Y_1) and (X_2, Y_2) correspond to consecutive pairs of variables of the collection PAIRS.
Arc input(s)	PAIRS
Arc generator	$PATH \mapsto \text{collection}(\text{pairs1}, \text{pairs2})$
Arc arity	2
Arc constraint(s)	$\text{pairs1.x CTRX pairs2.x} \vee \text{pairs1.y CTRY pairs2.y}$
Graph property(ies)	$NARC = NCHANGE$

Example

$$\text{change_pair} \left(3, \left\{ \begin{array}{l} x - 3 \quad y - 5, \\ x - 3 \quad y - 7, \\ x - 3 \quad y - 7, \\ x - 3 \quad y - 8, \\ x - 3 \quad y - 4, \\ x - 3 \quad y - 7, \\ x - 1 \quad y - 3, \\ x - 1 \quad y - 6, \\ x - 1 \quad y - 6, \\ x - 3 \quad y - 7 \end{array} \right\}, \neq, > \right)$$

In the previous example we have the following 3 changes:

- One change between pairs $x - 3 \quad y - 8$ and $x - 3 \quad y - 4$,
- One change between pairs $x - 3 \quad y - 7$ and $x - 1 \quad y - 3$,
- One change between pairs $x - 1 \quad y - 6$ and $x - 3 \quad y - 7$.

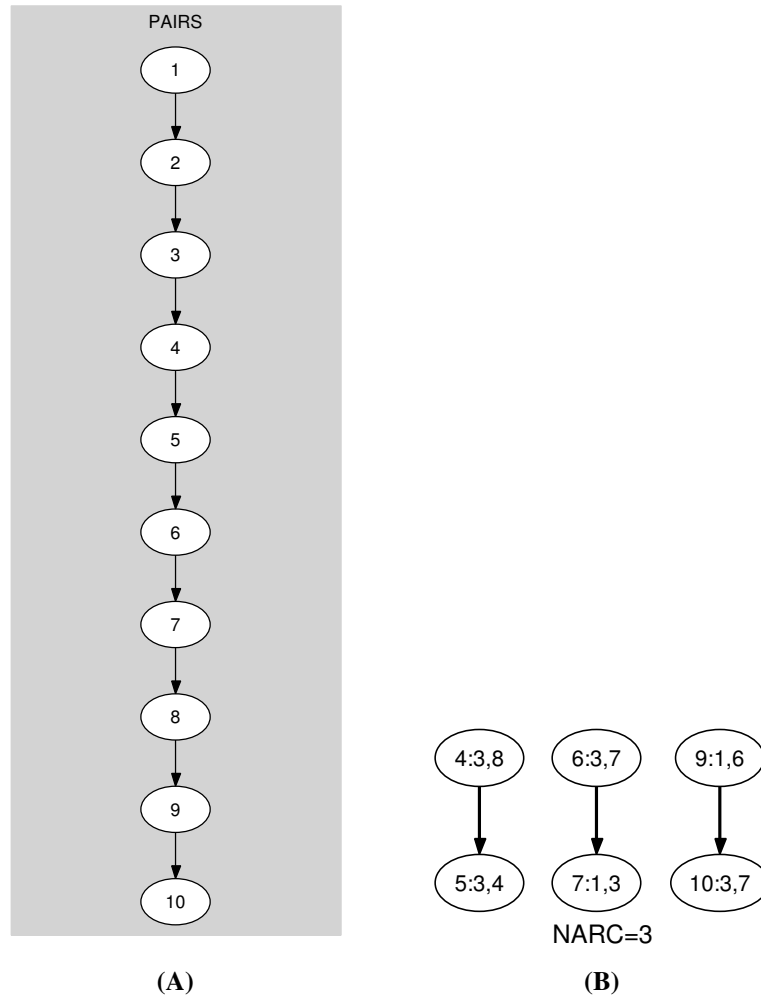


Figure 4.88: Initial and final graph of the change_pair constraint

Parts (A) and (B) of Figure 4.88 respectively show the initial and final graph. Since we use the **NARC** graph property, the arcs of the final graph are stressed in bold.

Graph model

Same as **change**, except that each item has two attributes **x** and **y**.

Automaton

Figure 4.89 depicts the automaton associated to the **change_pair** constraint. To each pair of consecutive pairs $((X_i, Y_i), (X_{i+1}, Y_{i+1}))$ of the collection **PAIRS** corresponds a 0-1 signature variable S_i . The following signature constraint links $X_i, Y_i, X_{i+1}, Y_{i+1}$ and S_i : $(X_i \text{ CTRX } X_{i+1}) \vee (Y_i \text{ CTRY } Y_{i+1}) \Leftrightarrow S_i$.

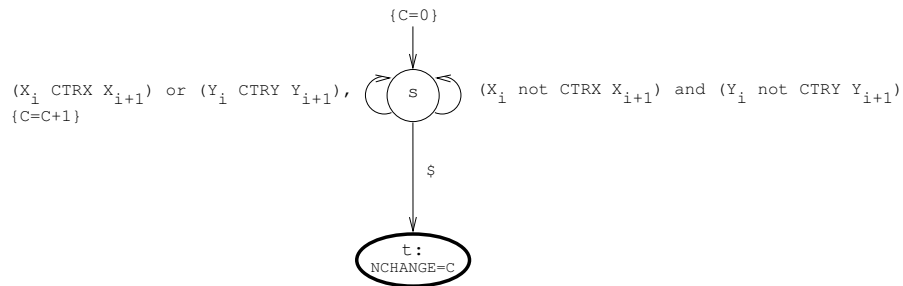


Figure 4.89: Automaton of the **change_pair** constraint

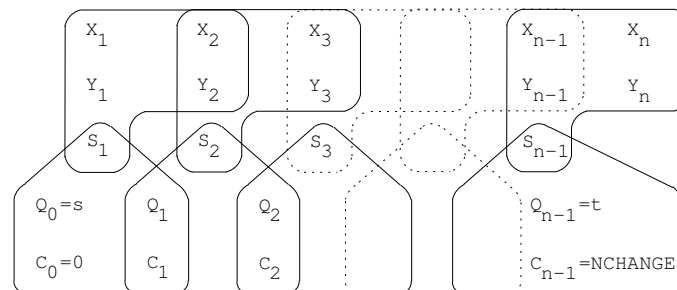


Figure 4.90: Hypergraph of the reformulation corresponding to the automaton of the **change_pair** constraint

Usage

Here is a typical example where this constraint is useful. Assume we have to produce a set of cables. A given quality and a given cross-section that respectively correspond to the **x** and **y** attributes of the previous pairs of variables characterize each cable. The problem is to sequence the different cables in order to minimize the number of times two consecutive wire cables C_1 and C_2 verify the following property: C_1 and C_2 do not have the same quality or the cross section of C_1 is greater than the cross section of C_2 .

See also

change.

Key words

timetabling constraint, number of changes, pair, automaton, automaton with counters, sliding cyclic(2) constraint network(2), acyclic, no_loop.

20030820

301

4.37 change_partition

Origin	Derived from change.
Constraint	change_partition(NCHANGE, VARIABLES, PARTITIONS)
Type(s)	VALUES : collection(val - int)
Argument(s)	NCHANGE : dvar VARIABLES : collection(var - dvar) PARTITIONS : collection(p - VALUES)
Restriction(s)	required(VALUES, val) distinct(VALUES, val) NCHANGE ≥ 0 NCHANGE < VARIABLES required(VARIABLES, var) required(PARTITIONS, p) PARTITIONS ≥ 2
Purpose	NCHANGE is the number of times that the following constraint holds: X and Y do not belong to the same partition of the collection PARTITIONS. X and Y correspond to consecutive variables of the collection VARIABLES.
Arc input(s)	VARIABLES
Arc generator	$PATH \mapsto \text{collection}(\text{variables1}, \text{variables2})$
Arc arity	2
Arc constraint(s)	in_same_partition(variables1.var, variables2.var, PARTITIONS)
Graph property(ies)	NARC = NCHANGE

Example	$\text{change_partition} \left(2, \left\{ \begin{array}{l} \text{var} - 6, \\ \text{var} - 6, \\ \text{var} - 2, \\ \text{var} - 1, \\ \text{var} - 3, \\ \text{var} - 3, \\ \text{var} - 1, \\ \text{var} - 6, \\ \text{var} - 2, \\ \text{var} - 2, \\ \text{var} - 2 \end{array} \right\}, \left\{ \begin{array}{l} p - \{\text{val} - 1, \text{val} - 3\}, \\ p - \{\text{val} - 4\}, \\ p - \{\text{val} - 2, \text{val} - 6\} \end{array} \right\} \right)$
----------------	---

In the previous example we have the following two changes:

- One change between values 2 and 1 (since 2 and 1 respectively belong to the third and the first partition),
- One change between values 1 and 6 (since 1 and 6 respectively belong to the first and the third partition).

Parts (A) and (B) of Figure 4.91 respectively show the initial and final graph. Since we use the NARC graph property, the arcs of the final graph are stressed in bold.

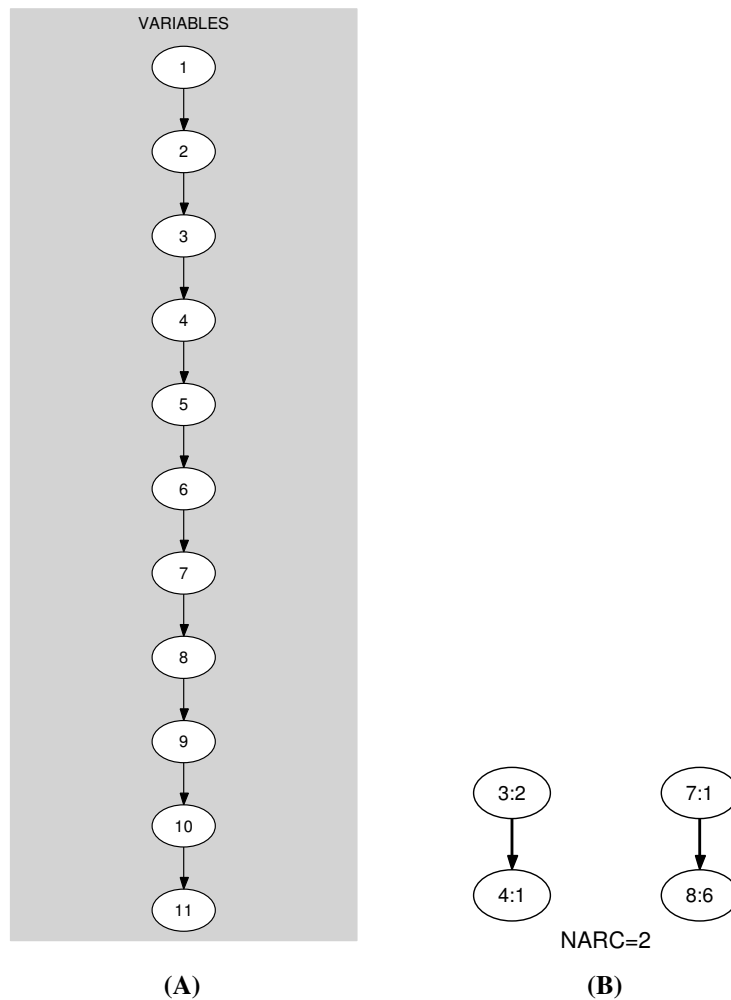


Figure 4.91: Initial and final graph of the `change_partition` constraint

Usage

This constraint is useful for the following problem: Assume you have to produce a set of orders, each order belonging to a given family. In the previous example we have three families that respectively correspond to values $\{1, 3\}$, to value $\{4\}$ and to values $\{2, 6\}$.

We would like to sequence the orders in such a way that we minimize the number of times two consecutive orders do not belong to the same family.

Algorithm

[65].

See also

`change`, `in_same_partition`.

Key words

timetabling constraint, number of changes, partition, acyclic, `no_loop`.

20000128

305

4.38 circuit

Origin	[2]
Constraint	<code>circuit(NODES)</code>
Synonym(s)	<code>atour</code> , <code>cycle</code> .
Argument(s)	<code>NODES</code> : <code>collection(index – int, succ – dvar)</code>
Restriction(s)	<code>required(NODES, [index, succ])</code> <code>NODES.index ≥ 1</code> <code>NODES.index ≤ NODES </code> <code>distinct(NODES, index)</code> <code>NODES.succ ≥ 1</code> <code>NODES.succ ≤ NODES </code>
Purpose	Enforce to cover a digraph G described by the <code>NODES</code> collection with one circuit visiting once all vertices of G .
<hr/>	
Arc input(s)	<code>NODES</code>
Arc generator	<code>CLIQUE</code> \mapsto <code>collection(nodes1, nodes2)</code>
Arc arity	2
Arc constraint(s)	<code>nodes1.succ = nodes2.index</code>
Graph property(ies)	<ul style="list-style-type: none"> • <code>MIN_NSCC = NODES </code> • <code>MAX_ID = 1</code>
<hr/>	
Example	$\text{circuit} \left(\left(\begin{array}{ll} \text{index} - 1 & \text{succ} - 2, \\ \text{index} - 2 & \text{succ} - 3, \\ \text{index} - 3 & \text{succ} - 4, \\ \text{index} - 4 & \text{succ} - 1 \end{array} \right) \right)$ <p>Parts (A) and (B) of Figure 4.92 respectively show the initial and final graph. The <code>circuit</code> constraint holds since the final graph consists of one circuit mentioning once every vertex of the initial graph.</p>
Graph model	The first graph property enforces to have one single strongly connected component containing <code> NODES </code> vertices. The second graph property imposes to only have circuits. Since each vertex of the final graph has only one successor we don't need to use set variables for representing the successors of a vertex.
Signature	Since the initial graph contains <code> NODES </code> vertices the final graph contains at most <code> NODES </code> vertices. Therefore we can rewrite the graph property <code>MIN_NSCC = NODES </code> to <code>MIN_NSCC ≥ NODES </code> . This leads to simplify <u>MIN_NSCC</u> to <u>MIN_NSCC</u> .

Because of the graph property $\text{MIN_NSCC} = |\text{NODES}|$ the final graph contains at least one vertex. Since a vertex v belongs to the final graph only if there is an arc that has v as one of its extremities the final graph contains at least one arc. Therefore MAX_ID is greater than or equal to 1. So we can rewrite the graph property $\text{MAX_ID} = 1$ to $\text{MAX_ID} \leq 1$. This leads to simplify MAX_ID to MAX_ID.

Remark

In the original `circuit` constraint of CHIP the `index` attribute was not explicitly present. It was implicitly defined as the position of a variable in a list.

Within the framework of linear programming [74] this constraint was introduced under the name `atour`. Within the KOALOG constraint system this constraint is called `cycle`.

Algorithm

Since all `succ` variables of the `NODES` collection have to take distinct values one can reuse the algorithms associated to the `alldifferent` constraint. A second necessary condition is to have no more than one strongly connected component. Further necessary conditions combining the fact that we have a perfect matching and one single strongly connected component can be found in [75]. When the graph is planar one can also use as a necessary condition discovered by Grinberg [76] for pruning.

See also

`cycle`, `tour`.

Key words

graph constraint, graph partitioning constraint, circuit, permutation, Hamiltonian, linear programming, `one_succ`.

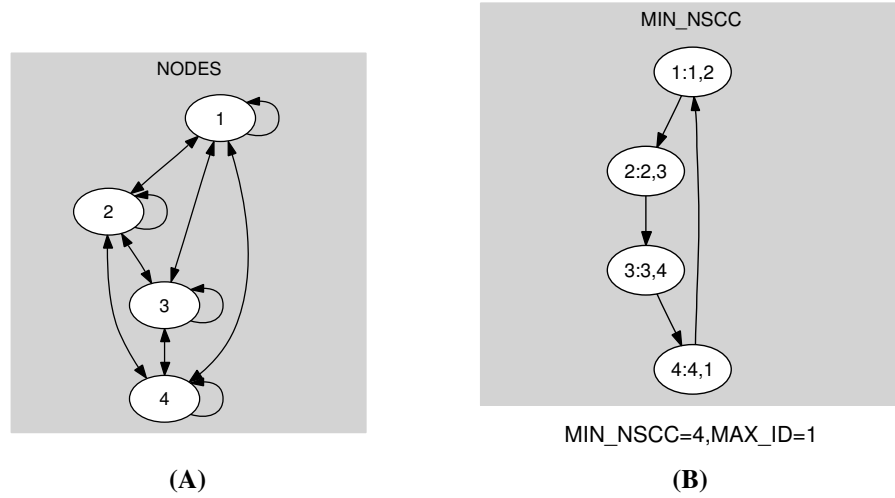


Figure 4.92: Initial and final graph of the circuit constraint

20030820

309

4.39 circuit_cluster

Origin	Inspired by [77].
Constraint	<code>circuit_cluster(NCIRCUIT, NODES)</code>
Argument(s)	NCIRCUIT : dvar NODES : collection(index – int, cluster – int, succ – dvar)
Restriction(s)	NCIRCUIT ≥ 1 NCIRCUIT $\leq \text{NODES} $ required(NODES, [index, cluster, succ]) NODES.index ≥ 1 NODES.index $\leq \text{NODES} $ distinct(NODES, index) NODES.succ ≥ 1 NODES.succ $\leq \text{NODES} $
Purpose	Consider a digraph G , described by the NODES collection, such that its vertices are partitioned among several clusters. NCIRCUIT is the number of circuits containing more than one vertex used for covering G in such a way that each cluster is visited by exactly one circuit of length greater than 1.
<hr/>	
Arc input(s)	NODES
Arc generator	<i>CLIQUE</i> \mapsto collection(nodes1, nodes2)
Arc arity	2
Arc constraint(s)	<ul style="list-style-type: none"> • nodes1.succ \neq nodes1.index • nodes1.succ = nodes2.index
Graph property(ies)	<ul style="list-style-type: none"> • NTREE = 0 • NSCC = NCIRCUIT
Sets	ALL_VERTICES \mapsto $\left[\text{variables} - \text{col} \left(\begin{array}{l} \text{VARIABLES} - \text{collection}(\text{var} - \text{dvar}), \\ [\text{item}(\text{var} - \text{NODES.cluster})] \end{array} \right) \right]$
Constraint(s) on sets	<ul style="list-style-type: none"> • alldifferent(variables) • nvalues(variables, =, size(NODES, cluster))

Example

$$\begin{array}{l}
 \text{circuit_cluster} \\
 \text{circuit_cluster}
 \end{array}
 \left(\begin{array}{l}
 1, \\
 2,
 \end{array} \left\{ \begin{array}{l}
 \left. \begin{array}{l}
 \text{index} - 1 \quad \text{cluster} - 1 \quad \text{succ} - 1, \\
 \text{index} - 2 \quad \text{cluster} - 1 \quad \text{succ} - 4, \\
 \text{index} - 3 \quad \text{cluster} - 2 \quad \text{succ} - 3, \\
 \text{index} - 4 \quad \text{cluster} - 2 \quad \text{succ} - 5, \\
 \text{index} - 5 \quad \text{cluster} - 3 \quad \text{succ} - 8, \\
 \text{index} - 6 \quad \text{cluster} - 3 \quad \text{succ} - 6, \\
 \text{index} - 7 \quad \text{cluster} - 3 \quad \text{succ} - 7, \\
 \text{index} - 8 \quad \text{cluster} - 4 \quad \text{succ} - 2, \\
 \text{index} - 9 \quad \text{cluster} - 4 \quad \text{succ} - 9
 \end{array} \right\} \\
 \left. \begin{array}{l}
 \text{index} - 1 \quad \text{cluster} - 1 \quad \text{succ} - 1, \\
 \text{index} - 2 \quad \text{cluster} - 1 \quad \text{succ} - 4, \\
 \text{index} - 3 \quad \text{cluster} - 2 \quad \text{succ} - 3, \\
 \text{index} - 4 \quad \text{cluster} - 2 \quad \text{succ} - 2, \\
 \text{index} - 5 \quad \text{cluster} - 3 \quad \text{succ} - 5, \\
 \text{index} - 6 \quad \text{cluster} - 3 \quad \text{succ} - 9, \\
 \text{index} - 7 \quad \text{cluster} - 3 \quad \text{succ} - 7, \\
 \text{index} - 8 \quad \text{cluster} - 4 \quad \text{succ} - 8, \\
 \text{index} - 9 \quad \text{cluster} - 4 \quad \text{succ} - 6
 \end{array} \right\}
 \end{array} \right)$$

Parts (A) and (B) of Figure 4.93 respectively show the initial and final graph associated to the second example. Since we use the NSCC graph property, we show the two strongly connected components of the final graph. They respectively correspond to the two circuits $2 \rightarrow 4 \rightarrow 2$ and $6 \rightarrow 9 \rightarrow 6$. Since all the vertices belongs to a circuit we have that **NTREE** = 0. The first example uses only one single circuit: $2 \rightarrow 4 \rightarrow 5 \rightarrow 8 \rightarrow 2$.

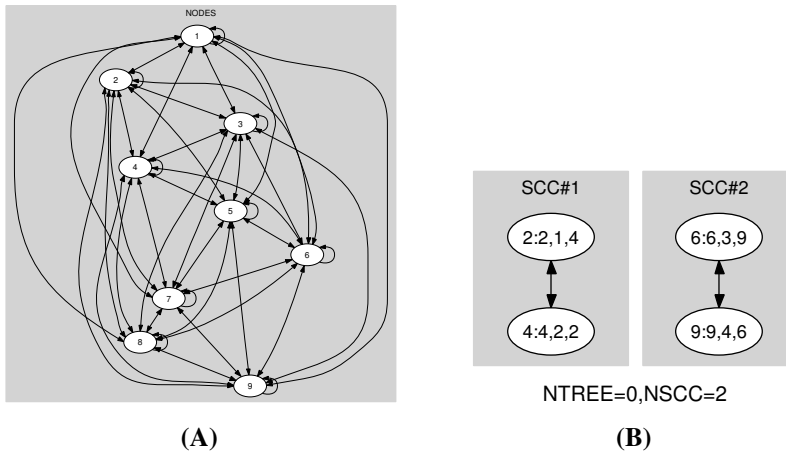


Figure 4.93: Initial and final graph of the circuit_cluster constraint

Graph model

In order to express the binary constraint linking two vertices one has to make explicit the identifier of each vertex as well as the cluster to which belong each vertex. This is why the circuit_cluster constraint considers objects that have the following three attributes:

- The attribute `index`, which is the identifier of a vertex.
- The attribute `cluster`, which is the cluster to which belong a vertex.

- The attribute `succ`, which is the unique successor of a vertex.

The partitioning of the clusters by different circuits is expressed in the following way:

- First observe the condition `nodes1.succ ≠ nodes1.index` prevents the final graph of containing any loop. Moreover the condition `nodes1.succ = nodes2.index` imposes no more than one successor for each vertex of the final graph.
- The graph property `NTREE = 0` enforces that all vertices of the final graph belong to one circuit.
- The graph property `NSCC = NCIRCUIT` express the fact that the number of strongly connected components of the final graph is equal to `NCIRCUIT`.
- The constraint `alldifferent(variables)` on the set `ALL_VERTICES` (i.e. all the vertices of the final graph) states that the cluster attributes of the vertices of the final graph should be pairwise distinct. This concretely means that no cluster should be visited more than once.
- The constraint `nvalues(variables, =, size(NODES, cluster))` on the set `ALL_VERTICES` conveys the fact that the number of distinct values of the cluster attribute of the vertices of the final graph should be equal to the total number of clusters. This implies that each cluster is visited at least one time.

Usage

A related abstraction in Operations Research was introduced in [77]. It was reported as the Generalized Travelling Salesman Problem (GTSP). The `circuit_cluster` constraint differs from the GTSP because of the two following points:

- Each node of our graph belongs to one single cluster,
- We do not constrain the number of circuits to be equal to one: the number of circuits should be equal to one of the values of the domain of the variable `NCIRCUIT`.

See also

`alldifferent`, `nvalues`.

Key words

graph constraint, connected component, cluster, `one_succ`.

20000128

313

4.40 circular_change

Origin	Derived from change.
Constraint	<code>circular_change(NCHANGE, VARIABLES, CTR)</code>
Argument(s)	<p>NCHANGE : dvar VARIABLES : collection(var – dvar) CTR : atom</p>
Restriction(s)	<p>NCHANGE ≥ 0 NCHANGE $\leq VARIABLES$ required(VARIABLES, var) CTR $\in [=, \neq, <, \geq, >, \leq]$</p>
Purpose	<p>NCHANGE is the number of times that CTR holds on consecutive variables of the collection VARIABLES. The last and the first variables of the collection VARIABLES are also considered to be consecutive.</p>
Arc input(s)	VARIABLES
Arc generator	<i>CIRCUIT</i> \mapsto collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	variables1.var CTR variables2.var
Graph property(ies)	NARC = NCHANGE
Example	$\text{circular_change} \left(4, \left\{ \begin{array}{l} \text{var} - 4, \\ \text{var} - 4, \\ \text{var} - 3, \\ \text{var} - 4, \\ \text{var} - 1 \end{array} \right\}, \neq \right)$ <p>In the previous example the changes are located between values 4 and 3, 3 and 4, 4 and 1, and 1 and 4. We count one change for each disequality constraint (between two consecutive variables) which holds. Parts (A) and (B) of Figure 4.94 respectively show the initial and final graph. Since we use the NARC graph property, the arcs of the final graph are stressed in bold.</p>
Graph model	Since we are also interested in the constraint that links the last and the first variable we use the arc generator <i>CIRCUIT</i> to produce the arcs of the initial graph.
Automaton	Figure 4.95 depicts the automaton associated to the <code>circular_change</code> constraint. To each pair of consecutive variables ($\text{VAR}_i, \text{VAR}_{(i \bmod VARIABLES)+1}$) of the collection VARIABLES corresponds a 0-1 signature variable S_i . The following signature constraint links $\text{VAR}_i, \text{VAR}_{(i \bmod VARIABLES)+1}$ and S_i : $\text{VAR}_i \text{ CTR } \text{VAR}_{(i \bmod VARIABLES)+1} \Leftrightarrow S_i$.

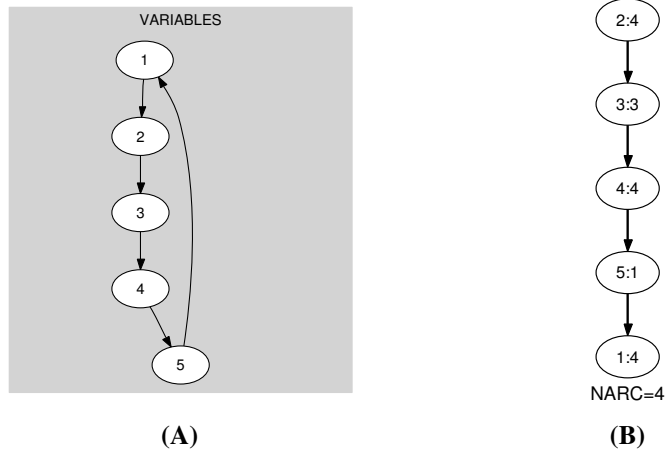


Figure 4.94: Initial and final graph of the circular_change constraint

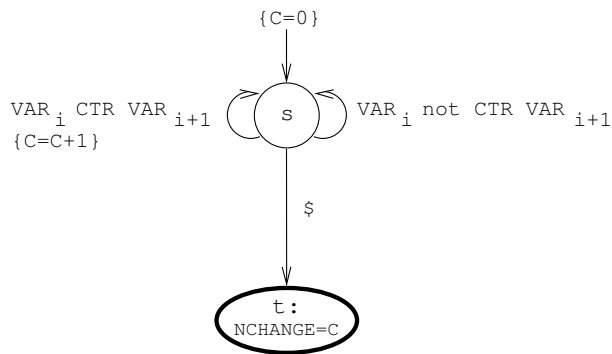


Figure 4.95: Automaton of the circular_change constraint

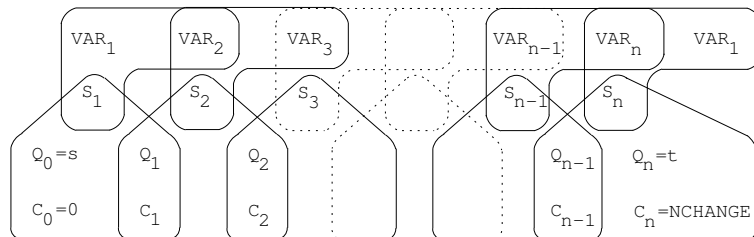


Figure 4.96: Hypergraph of the reformulation corresponding to the automaton of the circular_change constraint

See also change.

Key words timetabling constraint, number of changes, cyclic, automaton, automaton with counters, circular sliding cyclic(1) constraint network(2).

20030820

317

4.41 clique

Origin	[78]
Constraint	<code>clique(SIZE_CLIQU, NODES)</code>
Argument(s)	<code>SIZE_CLIQU</code> : dvar <code>NODES</code> : <code>collection(index - int, succ - svar)</code>
Restriction(s)	<code>SIZE_CLIQU</code> ≥ 0 <code>SIZE_CLIQU</code> $\leq \text{NODES} $ <code>required(NODES, [index, succ])</code> <code>NODES.index</code> ≥ 1 <code>NODES.index</code> $\leq \text{NODES} $ <code>distinct(NODES, index)</code>
Purpose	<p>Consider a digraph G described by the <code>NODES</code> collection: To the i^{th} item of the <code>NODES</code> collection corresponds the i^{th} vertex of G; To each value j of the i^{th} <code>succ</code> variable corresponds an arc from the i^{th} vertex to the j^{th} vertex. Select a subset \mathcal{S} of the vertices of G which forms a clique of size <code>SIZE_CLIQU</code> (i.e. there is an arc between each pair of distinct vertices of \mathcal{S}).</p>
Arc input(s)	<code>NODES</code>
Arc generator	<code>CLIQUE</code> (\neq) \mapsto <code>collection(nodes1, nodes2)</code>
Arc arity	2
Arc constraint(s)	<code>in_set(nodes2.index, nodes1.succ)</code>
Graph property(ies)	<ul style="list-style-type: none"> • <code>NARC</code> = <code>SIZE_CLIQU</code> * <code>SIZE_CLIQU</code> - <code>SIZE_CLIQU</code> • <code>NVERTEX</code> = <code>SIZE_CLIQU</code>
Example	$\text{clique} \left(3, \left\{ \begin{array}{l} \text{index} - 1 \quad \text{succ} - \emptyset, \\ \text{index} - 2 \quad \text{succ} - \{3, 5\}, \\ \text{index} - 3 \quad \text{succ} - \{2, 5\}, \\ \text{index} - 4 \quad \text{succ} - \emptyset, \\ \text{index} - 5 \quad \text{succ} - \{2, 3\} \end{array} \right\} \right)$ <p>Part (A) of Figure 4.97 shows the initial graph from which we start. It is derived from the set associated to each vertex. Each set describes the potential values of the <code>succ</code> attribute of a given vertex. Part (B) of Figure 4.97 gives the final graph associated to the example. Since we both use the <code>NARC</code> and <code>NVERTEX</code> graph properties, the arcs and the vertices of the final graph are stressed in bold. The final graph corresponds to a clique containing three vertices.</p>
Graph model	Observe the use of <i>set variables</i> for modelling the fact that the vertices of the final graph have more than one successor: The successor variable associated to each vertex contains the successors of the corresponding vertex.

Algorithm [78], [79].

See also `link_set_to_booleans`.

Key words graph constraint, maximum clique, constraint involving set variables.

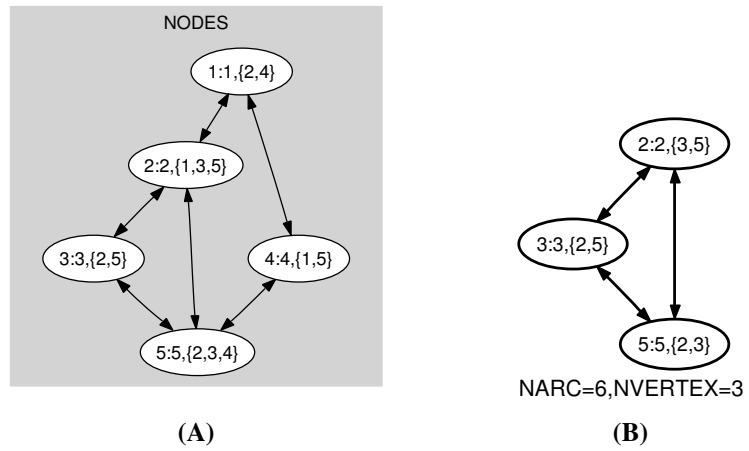


Figure 4.97: Initial and final graph of the clique set constraint

20030820

321

4.42 colored_matrix

Origin	KOALOG
Constraint	colored_matrix(C, L, K, MATRIX, CPROJ, LPROJ)
Synonym(s)	cardinality_matrix, card_matrix.
Argument(s)	<pre> C : int L : int K : int MATRIX : collection(column - int, line - int, var - dvar) CPROJ : collection(column - int, val - int, noccurrence - dvar) LPROJ : collection(line - int, val - int, noccurrence - dvar) </pre>
Restriction(s)	<pre> C ≥ 0 L ≥ 0 K ≥ 0 required(MATRIX, [column, line, var]) increasing_seq(MATRIX, [column, line]) MATRIX = C * L + C + L + 1 MATRIX.column ≥ 0 MATRIX.column ≤ C MATRIX.line ≥ 0 MATRIX.line ≤ L MATRIX.var ≥ 0 MATRIX.var ≤ K required(CPROJ, [column, val, noccurrence]) increasing_seq(CPROJ, [column, val]) CPROJ = C * K + C + K + 1 CPROJ.column ≥ 0 CPROJ.column ≤ C CPROJ.val ≥ 0 CPROJ.val ≤ K required(LPROJ, [line, val, noccurrence]) increasing_seq(LPROJ, [line, val]) LPROJ = L * K + L + K + 1 LPROJ.line ≥ 0 LPROJ.line ≤ L LPROJ.val ≥ 0 LPROJ.val ≤ K </pre>

Purpose

Given a matrix of domain variables, imposes a global_cardinality constraint involving cardinality variables on each column and each row of the matrix.

Example

colored_matrix

$$\left(\begin{array}{c} 1, 2, 4, \left\{ \begin{array}{l} \text{column} - 0 \quad \text{line} - 0 \quad \text{var} - 3, \\ \text{column} - 0 \quad \text{line} - 1 \quad \text{var} - 1, \\ \text{column} - 0 \quad \text{line} - 2 \quad \text{var} - 3, \\ \text{column} - 1 \quad \text{line} - 0 \quad \text{var} - 4, \\ \text{column} - 1 \quad \text{line} - 1 \quad \text{var} - 4, \\ \text{column} - 1 \quad \text{line} - 2 \quad \text{var} - 3 \end{array} \right\}, \\ \left\{ \begin{array}{l} \text{column} - 0 \quad \text{val} - 0 \quad \text{nocc} - 0, \\ \text{column} - 0 \quad \text{val} - 1 \quad \text{nocc} - 1, \\ \text{column} - 0 \quad \text{val} - 2 \quad \text{nocc} - 0, \\ \text{column} - 0 \quad \text{val} - 3 \quad \text{nocc} - 2, \\ \text{column} - 0 \quad \text{val} - 4 \quad \text{nocc} - 0, \\ \text{column} - 1 \quad \text{val} - 0 \quad \text{nocc} - 0, \\ \text{column} - 1 \quad \text{val} - 1 \quad \text{nocc} - 0, \\ \text{column} - 1 \quad \text{val} - 2 \quad \text{nocc} - 0, \\ \text{column} - 1 \quad \text{val} - 3 \quad \text{nocc} - 1, \\ \text{column} - 1 \quad \text{val} - 4 \quad \text{nocc} - 2 \end{array} \right\}, \\ \left\{ \begin{array}{l} \text{line} - 0 \quad \text{val} - 0 \quad \text{nocc} - 0, \\ \text{line} - 0 \quad \text{val} - 1 \quad \text{nocc} - 0, \\ \text{line} - 0 \quad \text{val} - 2 \quad \text{nocc} - 0, \\ \text{line} - 0 \quad \text{val} - 3 \quad \text{nocc} - 1, \\ \text{line} - 0 \quad \text{val} - 4 \quad \text{nocc} - 1, \\ \text{line} - 1 \quad \text{val} - 0 \quad \text{nocc} - 0, \\ \text{line} - 1 \quad \text{val} - 1 \quad \text{nocc} - 1, \\ \text{line} - 1 \quad \text{val} - 2 \quad \text{nocc} - 0, \\ \text{line} - 1 \quad \text{val} - 3 \quad \text{nocc} - 0, \\ \text{line} - 1 \quad \text{val} - 4 \quad \text{nocc} - 1, \\ \text{line} - 2 \quad \text{val} - 0 \quad \text{nocc} - 0, \\ \text{line} - 2 \quad \text{val} - 1 \quad \text{nocc} - 0, \\ \text{line} - 2 \quad \text{val} - 2 \quad \text{nocc} - 0, \\ \text{line} - 2 \quad \text{val} - 3 \quad \text{nocc} - 2, \\ \text{line} - 2 \quad \text{val} - 4 \quad \text{nocc} - 0 \end{array} \right\} \end{array} \right)$$

Remark

Within [80] the `colored_matrix` constraint is called `cardinality_matrix`.

Algorithm

The filtering algorithm described in [80] is based on network flow and does not achieve arc-consistency in general. However, when the number of values is restricted to two, the algorithm [80] achieves arc-consistency on the variables of the matrix. This corresponds in fact to a generalization of the problem called "Matrices composed of 0's and 1's" presented by Ford and Fulkerson [81].

See also

`global_cardinality`, `same`.

Key words

predefined constraint, timetabling constraint, matrix, matrix model.

4.43 coloured_cumulative

Origin	Derived from <i>cumulative</i> and <i>nvalues</i> .
Constraint	<i>coloured_cumulative</i> (TASKS, LIMIT)
Argument(s)	TASKS : collection(origin – dvar, duration – dvar, end – dvar, colour – dvar) LIMIT : int
Restriction(s)	require_at_least(2, TASKS, [origin, duration, end]) required(TASKS, colour) TASKS.duration ≥ 0 LIMIT ≥ 0
Purpose	Consider the set \mathcal{T} of tasks described by the TASKS collection. The <i>coloured_cumulative</i> constraint enforces that, at each point in time, the number of distinct colours of the set of tasks that overlap that point, does not exceed a given limit. For each task of \mathcal{T} it also imposes the constraint $\text{origin} + \text{duration} = \text{end}$.
<hr/>	
Arc input(s)	TASKS
Arc generator	<i>SELF</i> \mapsto collection(tasks)
Arc arity	1
Arc constraint(s)	tasks.origin + tasks.duration = tasks.end
Graph property(ies)	$\overline{\text{NARC}} = \text{TASKS} $
<hr/>	
Arc input(s)	TASKS TASKS
Arc generator	<i>PRODUCT</i> \mapsto collection(tasks1, tasks2)
Arc arity	2
Arc constraint(s)	<ul style="list-style-type: none"> • tasks1.duration > 0 • tasks2.origin ≤ tasks1.origin • tasks1.origin < tasks2.end
Sets	$\text{SUCC} \mapsto \left[\begin{array}{l} \text{source,} \\ \text{variables} - \text{col} \left(\begin{array}{l} \text{VARIABLES} - \text{collection}(\text{var} - \text{dvar}), \\ [\text{item}(\text{var} - \text{TASKS.colour})] \end{array} \right) \end{array} \right]$
Constraint(s) on sets	<i>nvalues</i> (variables, ≤, LIMIT)

Example

$$\text{coloured_cumulative} \left(\left(\begin{array}{cccc} \text{origin} - 1 & \text{duration} - 2 & \text{end} - 3 & \text{colour} - 1, \\ \text{origin} - 2 & \text{duration} - 9 & \text{end} - 11 & \text{colour} - 2, \\ \text{origin} - 3 & \text{duration} - 10 & \text{end} - 13 & \text{colour} - 3, \\ \text{origin} - 6 & \text{duration} - 6 & \text{end} - 12 & \text{colour} - 2, \\ \text{origin} - 7 & \text{duration} - 2 & \text{end} - 9 & \text{colour} - 3 \end{array} \right), 2 \right)$$

Parts (A) and (B) of Figure 4.98 respectively show the initial and final graph associated to the second graph constraint. On the one hand, each source vertex of the final graph can be interpreted as a time point. On the other hand the successors of a source vertex correspond to those tasks which overlap that time point. The `coloured_cumulative` constraint holds since for each successor set \mathcal{S} of the final graph the number of distinct colours of the tasks in \mathcal{S} does not exceed the LIMIT 2. Figure 4.99 shows the solution associated to the previous example.

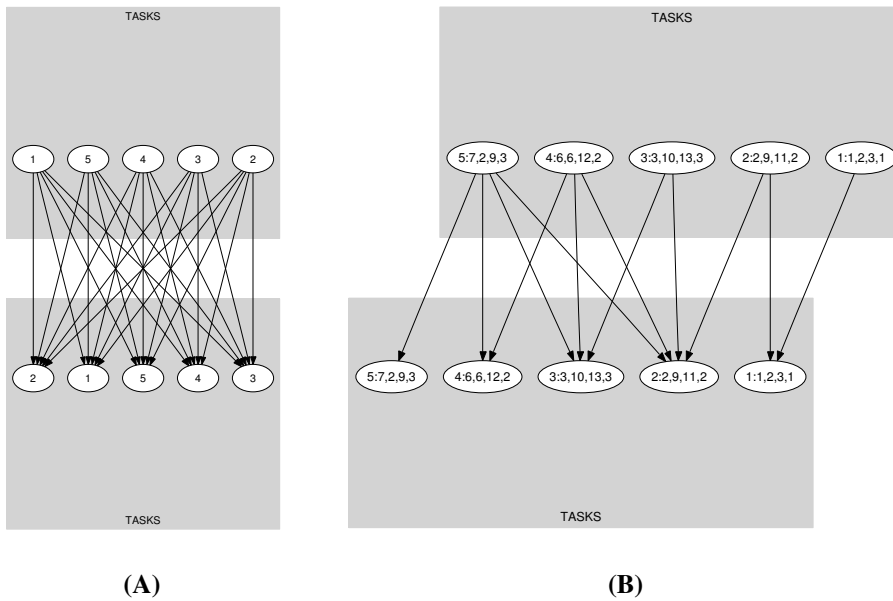


Figure 4.98: Initial and final graph of the `coloured_cumulative` constraint

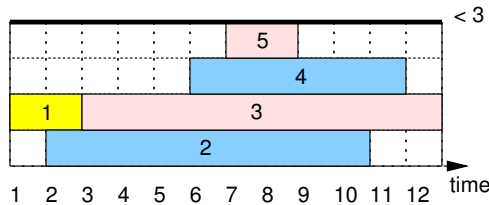


Figure 4.99: A coloured cumulative solution with at most two distinct colours in parallel

Graph model

Same as `cumulative`, except that we use an other constraint for computing the resource

consumption at each time point.

Signature

Since *TASKS* is the maximum number of vertices of the final graph of the first graph constraint we can rewrite $\text{NARC} = |\text{TASKS}|$ to $\text{NARC} \geq |\text{TASKS}|$. This leads to simplify $\overline{\text{NARC}}$ to $\overline{\text{NARC}}$.

Usage

Useful for scheduling problems where a machine can only proceed in parallel a maximum number of tasks of distinct type. This condition cannot be modelled by the classical cumulative constraint.

See also

coloured_cumulatives, *cumulative*, *nvalues*.

Key words

scheduling constraint, resource constraint, temporal constraint, coloured, number of distinct values.

20000128

327

4.44 coloured_cumulatives

Origin	Derived from cumulatives and nvalues.
Constraint	coloured_cumulatives(TASKS, MACHINES)
Argument(s)	$\text{TASKS} \quad : \quad \text{collection} \left(\begin{array}{l} \text{machine} - \text{dvar}, \\ \text{origin} - \text{dvar}, \\ \text{duration} - \text{dvar}, \\ \text{end} - \text{dvar}, \\ \text{colour} - \text{dvar} \end{array} \right)$ $\text{MACHINES} \quad : \quad \text{collection}(\text{id} - \text{int}, \text{capacity} - \text{int})$
Restriction(s)	required(TASKS, [machine, colour]) require_at_least(2, TASKS, [origin, duration, end]) TASKS.duration \geq 0 required(MACHINES, [id, capacity]) distinct(MACHINES, id) MACHINES.capacity \geq 0

Purpose

Consider a set \mathcal{T} of tasks described by the TASKS collection. The coloured_cumulatives constraint enforces for each machine m of the MACHINES collection the following condition: At each point in time p , the numbers of distinct colours of the set of tasks that both overlap that point p and are assigned to machine m does not exceed the capacity of machine m . It also imposes for each task of \mathcal{T} the constraint $\text{origin} + \text{duration} = \text{end}$.

Arc input(s)	TASKS
Arc generator	$\text{SELF} \mapsto \text{collection}(\text{tasks})$
Arc arity	1
Arc constraint(s)	tasks.origin + tasks.duration = tasks.end
Graph property(ies)	$\text{NARC} = \text{TASKS} $

For all items of MACHINES:

Arc input(s)	TASKS TASKS
Arc generator	$\text{PRODUCT} \mapsto \text{collection}(\text{tasks1}, \text{tasks2})$
Arc arity	2
Arc constraint(s)	<ul style="list-style-type: none"> • tasks1.machine = MACHINES.id • tasks1.machine = tasks2.machine • tasks1.duration > 0 • tasks2.origin \leq tasks1.origin • tasks1.origin < tasks2.end

Sets $SUCC \mapsto$

$$\left[\begin{array}{l} \text{source,} \\ \text{variables - col} \left(\begin{array}{l} \text{VARIABLES - collection}(\text{var - dvar}), \\ [\text{item}(\text{var - TASKS.colour})] \end{array} \right) \end{array} \right]$$

Constraint(s) on sets $nvalues(\text{variables}, \leq, \text{MACHINES.capacity})$

Example $\text{coloured_cumulatives} \left(\left\{ \begin{array}{l} \text{machine - 1 origin - 6 duration - 6 end - 12 colour - 1,} \\ \text{machine - 1 origin - 2 duration - 9 end - 11 colour - 2,} \\ \text{machine - 2 origin - 7 duration - 3 end - 10 colour - 2,} \\ \text{machine - 1 origin - 1 duration - 2 end - 3 colour - 1,} \\ \text{machine - 2 origin - 4 duration - 5 end - 9 colour - 2,} \\ \text{machine - 1 origin - 3 duration - 10 end - 13 colour - 1} \end{array} \right\}, \left\{ \begin{array}{l} \text{id - 1 capacity - 2,} \\ \text{id - 2 capacity - 1} \end{array} \right\} \right)$

Parts (A) and (B) of Figure 4.100 respectively shows the initial and final graph associated to machines 1 and 2. On the one hand, each source vertex of the final graph can be interpreted as a time point p on a specific machine m . On the other hand the successors of a source vertex correspond to those tasks which both overlap that time point p and are assigned to machine m . The `coloured_cumulatives` constraint holds since for each successor set \mathcal{S} of the final graph the number of distinct colours in \mathcal{S} does not exceed the capacity of the machine corresponding to the time point associated to \mathcal{S} . Figure 4.101 shows the solution associated to the previous example. For machine 1 we have at most two distinct colours in parallel, while for machine 2 we have no more than one single colour in parallel.

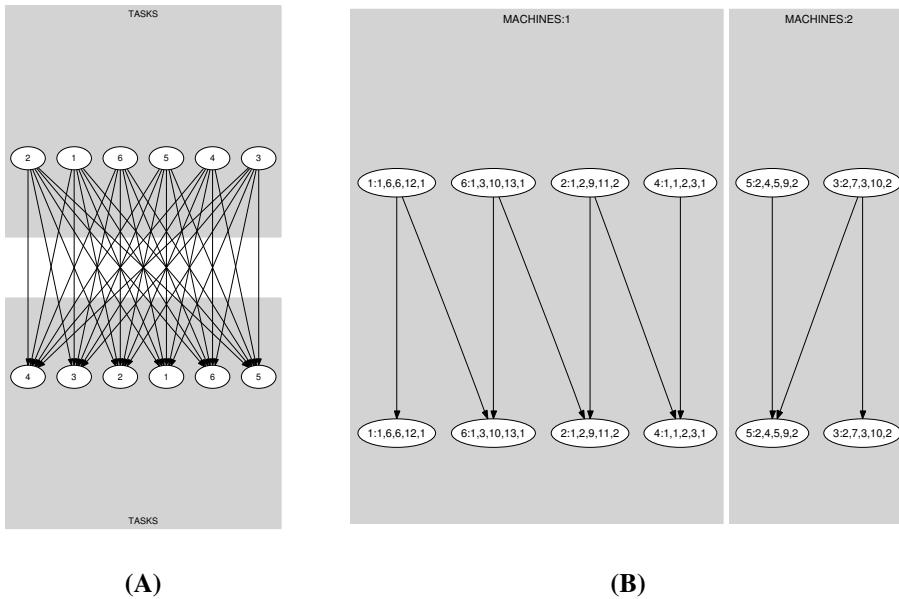


Figure 4.100: Initial and final graph of the `coloured_cumulatives` constraint

Signature	Since TASKS is the maximum number of vertices of the final graph of the first graph constraint we can rewrite $\text{NARC} = \text{TASKS} $ to $\text{NARC} \geq \text{TASKS} $. This leads to simplify $\overline{\text{NARC}}$ to $\overline{\text{NARC}}$.
Usage	Useful for scheduling problems where several machines are available and where you have to assign each task to a specific machine. In addition each machine can only proceed in parallel a maximum number of tasks of distinct types.
See also	<code>coloured_cumulative</code> , <code>cumulative</code> , <code>cumulatives</code> , <code>nvalues</code> .
Key words	scheduling constraint, resource constraint, temporal constraint, coloured, number of distinct values.

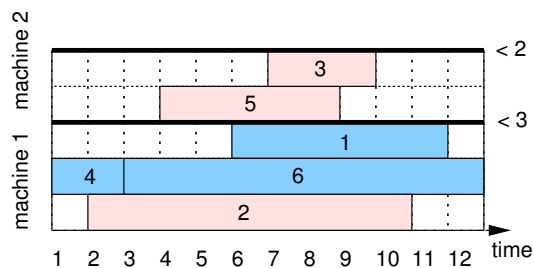


Figure 4.101: Assignment of the tasks on the two machines

20000128

331

4.45 common

Origin	N. Beldiceanu
Constraint	<code>common(NCOMMON1, NCOMMON2, VARIABLES1, VARIABLES2)</code>
Argument(s)	<pre> NCOMMON1 : dvar NCOMMON2 : dvar VARIABLES1 : collection(var - dvar) VARIABLES2 : collection(var - dvar) </pre>
Restriction(s)	<pre> NCOMMON1 ≥ 0 NCOMMON1 ≤ VARIABLES1 NCOMMON2 ≥ 0 NCOMMON2 ≤ VARIABLES2 required(VARIABLES1, var) required(VARIABLES2, var) </pre>

Purpose	<p>NCOMMON1 is the number of variables of the collection of variables VARIABLES1 taking a value in VARIABLES2. NCOMMON2 is the number of variables of the collection of variables VARIABLES2 taking a value in VARIABLES1.</p>
----------------	---

Arc input(s)	VARIABLES1 VARIABLES2
Arc generator	<i>PRODUCT</i> \mapsto <code>collection(variables1, variables2)</code>
Arc arity	2
Arc constraint(s)	<code>variables1.var = variables2.var</code>
Graph property(ies)	<ul style="list-style-type: none"> • NSOURCE = NCOMMON1 • NSINK = NCOMMON2

Example	$\text{common} \left(3, 4, \left\{ \begin{array}{l} \text{var} - 1, \text{var} - 9, \text{var} - 1, \text{var} - 5, \\ \left\{ \begin{array}{l} \text{var} - 2, \\ \text{var} - 1, \\ \text{var} - 9, \\ \text{var} - 9, \\ \text{var} - 6, \\ \text{var} - 9 \end{array} \right\} \end{array} \right. \right)$
----------------	--

Parts (A) and (B) of Figure 4.102 respectively show the initial and final graph. Since we use the **NSOURCE** and **NSINK** graph properties, the source and sink vertices of the final graph are stressed with a double circle. Since the final graph has only 3 sources and 4 sinks the variables NCOMMON1 and NCOMMON2 are respectively equal to 3 and 4. Note that all the vertices corresponding to the variables that take values 5, 2 or 6 were removed from the final graph since there is no arc for which the associated equality constraint holds.

20000128

333

See also `alldifferent_on_intersection`, `nvalue_on_intersection`, `same_intersection`.

Key words constraint between two collections of variables, acyclic, bipartite, no_loop.

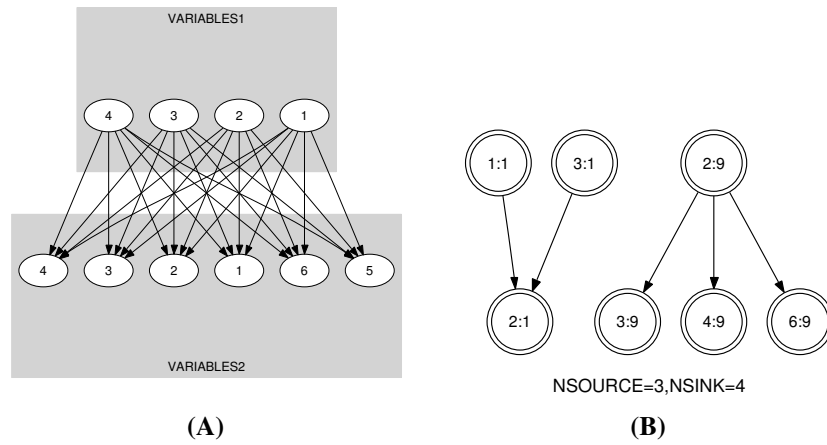


Figure 4.102: Initial and final graph of the common constraint

20000128

335

4.46 common_interval

Origin	Derived from common.
Constraint	<code>common_interval(NCOMMON1, NCOMMON2, VARIABLES1, VARIABLES2, SIZE_INTERVAL)</code>
Argument(s)	<pre> NCOMMON1 : dvar NCOMMON2 : dvar VARIABLES1 : collection(var - dvar) VARIABLES2 : collection(var - dvar) SIZE_INTERVAL : int </pre>
Restriction(s)	<pre> NCOMMON1 ≥ 0 NCOMMON1 ≤ VARIABLES1 NCOMMON2 ≥ 0 NCOMMON2 ≤ VARIABLES2 required(VARIABLES1, var) required(VARIABLES2, var) SIZE_INTERVAL > 0 </pre>
Purpose	<div style="border: 1px solid black; padding: 5px;"> <p>NCOMMON1 is the number of variables of the collection of variables VARIABLES1 taking a value in one of the intervals derived from the values assigned to the variables of the collection VARIABLES2: To each value v assigned to a variable of the collection VARIABLES2 we associate the interval $[SIZE_INTERVAL \cdot \lfloor v/SIZE_INTERVAL \rfloor, SIZE_INTERVAL \cdot \lfloor v/SIZE_INTERVAL \rfloor + SIZE_INTERVAL - 1]$.</p> <p>NCOMMON2 is the number of variables of the collection of variables VARIABLES2 taking a value in one of the intervals derived from the values assigned to the variables of the collection VARIABLES1: To each value v assigned to a variable of the collection VARIABLES1 we associate the interval $[SIZE_INTERVAL \cdot \lfloor v/SIZE_INTERVAL \rfloor, SIZE_INTERVAL \cdot \lfloor v/SIZE_INTERVAL \rfloor + SIZE_INTERVAL - 1]$.</p> </div>
Arc input(s)	VARIABLES1 VARIABLES2
Arc generator	$PRODUCT \mapsto \text{collection}(\text{variables1}, \text{variables2})$
Arc arity	2
Arc constraint(s)	$\text{variables1.var}/SIZE_INTERVAL = \text{variables2.var}/SIZE_INTERVAL$
Graph property(ies)	<ul style="list-style-type: none"> • NSOURCE = NCOMMON1 • NSINK = NCOMMON2

Example

$$\text{common_interval} \left(\begin{array}{l} 3, 2, \left\{ \begin{array}{l} \text{var} - 8, \\ \text{var} - 6, \\ \text{var} - 6, \\ \text{var} - 0 \end{array} \right\}, \\ \left\{ \begin{array}{l} \text{var} - 7, \\ \text{var} - 3, \\ \text{var} - 3, \\ \text{var} - 3, \\ \text{var} - 3, \\ \text{var} - 7 \end{array} \right\}, 3 \end{array} \right)$$

In the previous example, the last parameter `SIZE_INTERVAL` defines the following family of intervals $[3 \cdot k, 3 \cdot k + 2]$, where k is an integer. Parts (A) and (B) of Figure 4.103 respectively show the initial and final graph. Since we use the `NSOURCE` and `NSINK` graph properties, the source and sink vertices of the final graph are stressed with a double circle. Since the graph has only 3 sources and 2 sinks the variables `NCOMMON1` and `NCOMMON2` are respectively equal to 3 and 2. Note that the vertices corresponding to the variables that take values 0 or 3 were removed from the final graph since there is no arc for which the associated arc constraint holds.

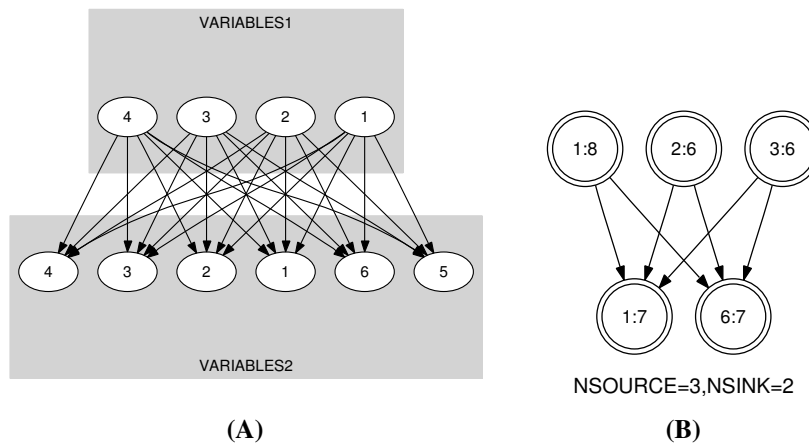


Figure 4.103: Initial and final graph of the `common_interval` constraint

See also `common`.

Key words constraint between two collections of variables, interval, acyclic, bipartite, no_loop.

4.47 common_modulo

Origin	Derived from common.
Constraint	<code>common_modulo(NCOMMON1, NCOMMON2, VARIABLES1, VARIABLES2, M)</code>
Argument(s)	<pre> NCOMMON1 : dvar NCOMMON2 : dvar VARIABLES1 : collection(var - dvar) VARIABLES2 : collection(var - dvar) M : int </pre>
Restriction(s)	<pre> NCOMMON1 ≥ 0 NCOMMON1 ≤ VARIABLES1 NCOMMON2 ≥ 0 NCOMMON2 ≤ VARIABLES2 required(VARIABLES1, var) required(VARIABLES2, var) M > 0 </pre>
Purpose	<p>NCOMMON1 is the number of variables of the collection of variables VARIABLES1 taking a value situated in an equivalence class (congruence modulo a fixed number M) derived from the values assigned to the variables of VARIABLES2 and from M.</p> <p>NCOMMON2 is the number of variables of the collection of variables VARIABLES2 taking a value situated in an equivalence class (congruence modulo a fixed number M) derived from the values assigned to the variables of VARIABLES1 and from M.</p>
Arc input(s)	VARIABLES1 VARIABLES2
Arc generator	<i>PRODUCT</i> \mapsto <code>collection(variables1, variables2)</code>
Arc arity	2
Arc constraint(s)	<code>variables1.var mod M = variables2.var mod M</code>
Graph property(ies)	<ul style="list-style-type: none"> • NSOURCE = NCOMMON1 • NSINK = NCOMMON2

Example

$$\text{common_modulo} \left(3, 4, \left\{ \begin{array}{l} \text{var} - 0, \text{var} - 4, \text{var} - 0, \text{var} - 8, \\ \text{var} - 7, \\ \text{var} - 5, \\ \text{var} - 4, \\ \text{var} - 9, \\ \text{var} - 2, \\ \text{var} - 4 \end{array} \right\}, 5 \right)$$

Parts (A) and (B) of Figure 4.104 respectively show the initial and final graph. Since we use the **NSOURCE** and **NSINK** graph properties, the source and sink vertices of the final graph are stressed with a double circle. Since the graph has only 3

sources and 4 sinks the variables NCOMMON1 and NCOMMON2 are respectively equal to 3 and 4. Note that the vertices corresponding to the variables that take values 8, 7 or 2 were removed from the final graph since there is no arc for which the associated arc constraint holds.

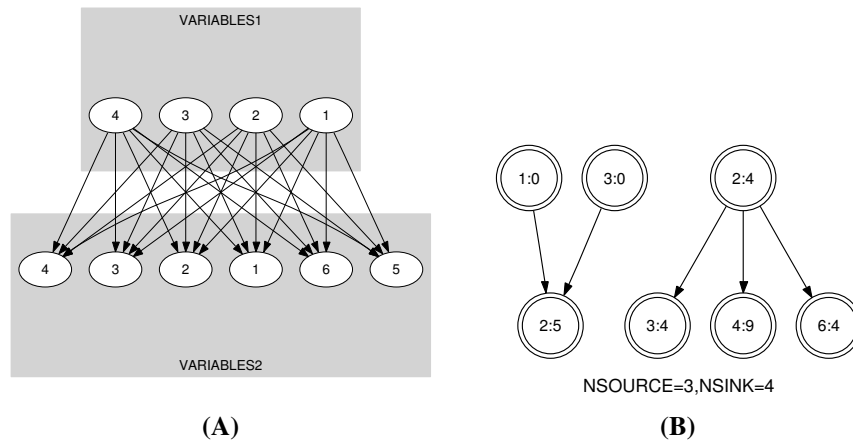


Figure 4.104: Initial and final graph of the common_modulo constraint

See also common.

Key words constraint between two collections of variables, modulo, acyclic, bipartite, no_loop.

4.48 common_partition

Origin	Derived from <code>common</code> .
Constraint	<code>common_partition(NCOMMON1, NCOMMON2, VARIABLES1, VARIABLES2, PARTITIONS)</code>
Type(s)	<code>VALUES : collection(val – int)</code>
Argument(s)	<code>NCOMMON1 : dvar</code> <code>NCOMMON2 : dvar</code> <code>VARIABLES1 : collection(var – dvar)</code> <code>VARIABLES2 : collection(var – dvar)</code> <code>PARTITIONS : collection(p – VALUES)</code>
Restriction(s)	<code>required(VALUES, val)</code> <code>distinct(VALUES, val)</code> <code>NCOMMON1 ≥ 0</code> <code>NCOMMON1 ≤ VARIABLES1 </code> <code>NCOMMON2 ≥ 0</code> <code>NCOMMON2 ≤ VARIABLES2 </code> <code>required(VARIABLES1, var)</code> <code>required(VARIABLES2, var)</code> <code>required(PARTITIONS, p)</code> <code> PARTITIONS ≥ 2</code>
Purpose	<p><code>NCOMMON1</code> is the number of variables of the <code>VARIABLES1</code> collection taking a value in a partition derived from the values assigned to the variables of <code>VARIABLES2</code> and from <code>PARTITIONS</code>. <code>NCOMMON2</code> is the number of variables of the <code>VARIABLES2</code> collection taking a value in a partition derived from the values assigned to the variables of <code>VARIABLES1</code> and from <code>PARTITIONS</code>.</p>
Arc input(s)	<code>VARIABLES1 VARIABLES2</code>
Arc generator	<code>PRODUCT</code> \mapsto <code>collection(variables1, variables2)</code>
Arc arity	2
Arc constraint(s)	<code>in_same_partition(variables1.var, variables2.var, PARTITIONS)</code>
Graph property(ies)	<ul style="list-style-type: none"> • <code>NSOURCE = NCOMMON1</code> • <code>NSINK = NCOMMON2</code>

Example

$$\text{common_partition} \left(\begin{array}{l} 3, 4, \left\{ \begin{array}{l} \text{var} - 2, \\ \text{var} - 3, \\ \text{var} - 6, \\ \text{var} - 0 \end{array} \right\}, \\ \left\{ \begin{array}{l} \text{var} - 0, \\ \text{var} - 6, \\ \text{var} - 3, \\ \text{var} - 3, \\ \text{var} - 7, \\ \text{var} - 1 \end{array} \right\}, \\ \left\{ \begin{array}{l} p - \{\text{val} - 1, \text{val} - 3\}, \\ p - \{\text{val} - 4\}, \\ p - \{\text{val} - 2, \text{val} - 6\} \end{array} \right\} \end{array} \right)$$

Parts (A) and (B) of Figure 4.105 respectively show the initial and final graph. Since we use the **NSOURCE** and **NSINK** graph properties, the source and sink vertices of the final graph are stressed with a double circle. Since the graph has only 3 sources and 4 sinks the variables **NCOMMON1** and **NCOMMON2** are respectively equal to 3 and 4. Note that the vertices corresponding to the variables that take values 0 or 7 were removed from the final graph since there is no arc for which the associated **in_same_partition** constraint holds.

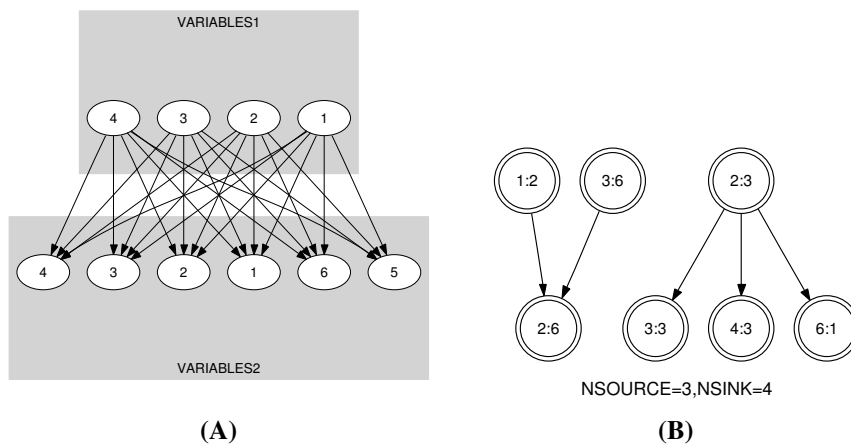


Figure 4.105: Initial and final graph of the **common_partition** constraint

See also `common`, `in_same_partition`.

Key words constraint between two collections of variables, partition, acyclic, bipartite, no_loop.

4.49 connect_points

Origin	N. Beldiceanu
Constraint	<code>connect_points(SIZE1, SIZE2, SIZE3, NGROUP, POINTS)</code>
Argument(s)	<pre> SIZE1 : int SIZE2 : int SIZE3 : int NGROUP : dvar POINTS : collection(p - dvar) </pre>
Restriction(s)	<pre> SIZE1 > 0 SIZE2 > 0 SIZE3 > 0 NGROUP ≥ 0 NGROUP ≤ POINTS SIZE1 * SIZE2 * SIZE3 = POINTS required(POINTS, p) </pre>
Purpose	<p>On a 3-dimensional grid of variables, number of groups, where a group consists of a connected set of variables which all have a same value distinct from 0.</p>
<hr/>	
Arc input(s)	POINTS
Arc generator	$GRID([SIZE1, SIZE2, SIZE3]) \mapsto \text{collection}(\text{points1}, \text{points2})$
Arc arity	2
Arc constraint(s)	<ul style="list-style-type: none"> • $\text{points1.p} \neq 0$ • $\text{points1.p} = \text{points2.p}$
Graph property(ies)	$NSCC = NGROUP$

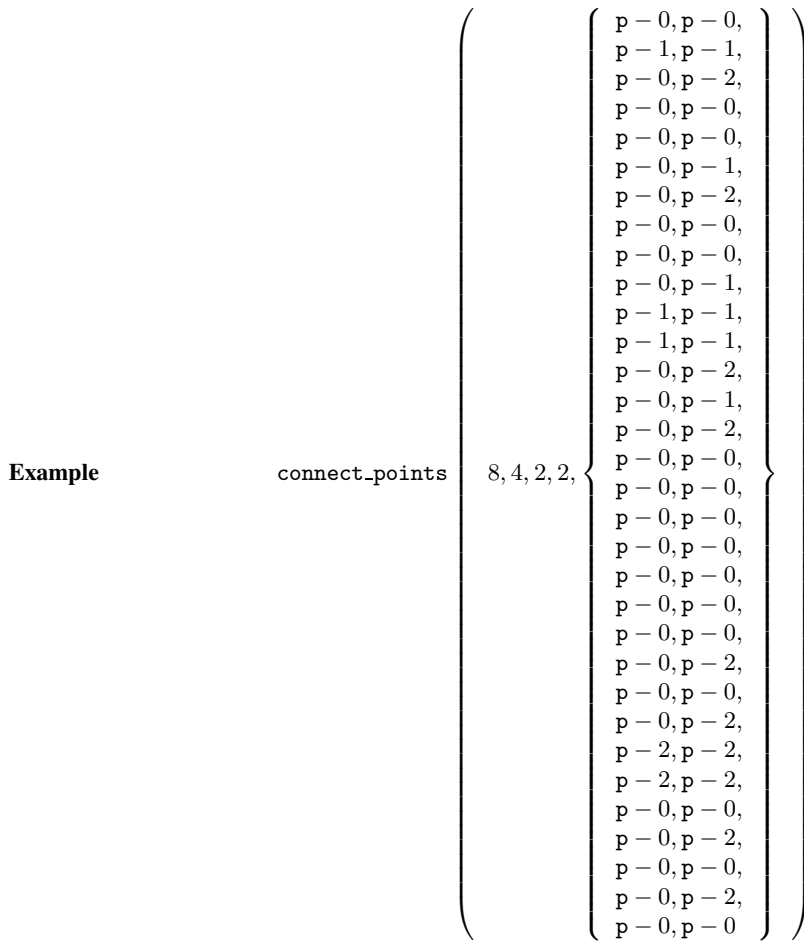


Figure 4.106 gives the initial graph constructed by the *GRID* arc generator. Figure 4.107 corresponds to the solution where we describe separately each layer of the grid. We have two groups: A first one for the variables assigned to value 1, and a second one for the variables assigned to value 2.

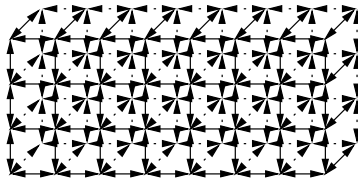


Figure 4.106: Graph generated by *GRID* ([8, 4, 2])

Usage Wiring problems [82], [83].

Key words geometrical constraint, channel routing, strongly connected component, joker value,

symmetric.

0	0	1	1	0	2	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	2	0	0	0	0	0	0	0	2	0	0
0	0	0	1	1	1	1	1	0	2	2	2	2	2	0	0
0	2	0	1	0	2	0	0	0	2	0	0	0	2	0	0

Figure 4.107: The two layers of the solution

20000128

345

4.50 correspondence

Origin	Derived from <code>sort_permutation</code> by removing the sorting condition.
Constraint	<code>correspondence(FROM, PERMUTATION, TO)</code>
Argument(s)	<pre>FROM : collection(fvar - dvar) PERMUTATION : collection(var - dvar) TO : collection(tvar - dvar)</pre>
Restriction(s)	<pre> PERMUTATION = FROM PERMUTATION = TO PERMUTATION.var ≥ 1 PERMUTATION.var ≤ PERMUTATION alldifferent(PERMUTATION) required(FROM, fvar) required(PERMUTATION, var) required(TO, tvar)</pre>
Purpose	<p>The variables of the TO collection correspond to the variables of the FROM collection according to the permutation expressed by PERMUTATION.</p>
Derived Collection(s)	<hr/> $\text{col} \left(\begin{array}{l} \text{FROM_PERMUTATION} - \text{collection}(\text{fvar} - \text{dvar}, \text{var} - \text{dvar}), \\ [\text{item}(\text{fvar} - \text{FROM.fvar}, \text{var} - \text{PERMUTATION.var})] \end{array} \right)$ <hr/>
Arc input(s)	FROM_PERMUTATION TO
Arc generator	<i>PRODUCT</i> \mapsto <code>collection(from_permutation, to)</code>
Arc arity	2
Arc constraint(s)	<ul style="list-style-type: none"> • <code>from_permutation.fvar = to.tvar</code> • <code>from_permutation.var = to.key</code>
Graph property(ies)	<hr/> <code>NARC = PERMUTATION </code> <hr/>

Example

$$\text{correspondence} \left(\left(\left. \begin{array}{l} \text{fvar} - 1, \\ \text{fvar} - 9, \\ \text{fvar} - 1, \\ \text{fvar} - 5, \\ \text{fvar} - 2, \\ \text{fvar} - 1 \end{array} \right\}, \left. \begin{array}{l} \text{var} - 6, \\ \text{var} - 1, \\ \text{var} - 3, \\ \text{var} - 5, \\ \text{var} - 4, \\ \text{var} - 2 \end{array} \right\}, \left. \begin{array}{l} \text{tvar} - 9, \\ \text{tvar} - 1, \\ \text{tvar} - 1, \\ \text{tvar} - 2, \\ \text{tvar} - 5, \\ \text{tvar} - 1 \end{array} \right\} \right)$$

Parts (A) and (B) of Figure 4.108 respectively show the initial and final graph. In both graphs the source vertices correspond to the derived collection FROM_PERMUTATION, while the sink vertices correspond to the collection TO. Since the final graph contains exactly |PERMUTATION| arcs the correspondence constraint holds. As we use the NARC graph property, the arcs of the final graph are stressed in bold.

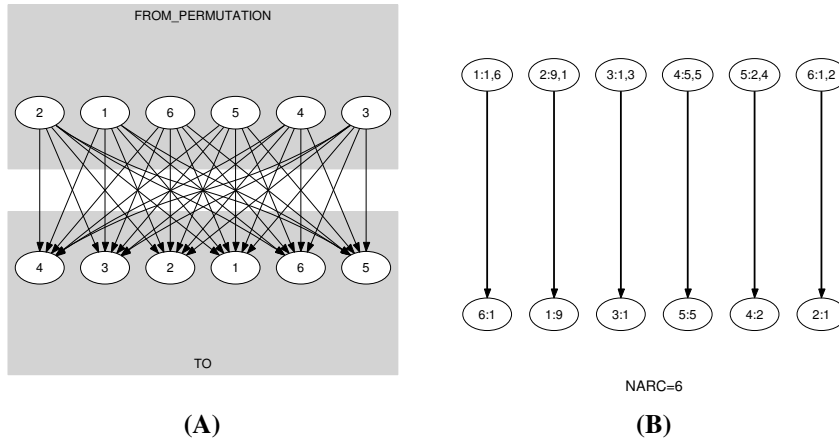


Figure 4.108: Initial and final graph of the correspondence constraint

Signature

Because of the second condition `from_permutation.var = to.key` of the arc constraint and since both, the `var` attributes of the collection `FROM_PERMUTATION` and the `key` attributes of the collection `TO` are all distinct, the final graph contains at most |PERMUTATION| arcs. Therefore we can rewrite the graph property `NARC = |PERMUTATION|` to `NARC ≥ |PERMUTATION|`. This leads to simplify NARC to NARC.

Remark

Similar to the same constraint except that we also provide the permutation which allows to

go from the items of collection FROM to the items of collection TO.

See also

same, sort_permutation.

Key words

constraint between three collections of variables, permutation, derived collection, acyclic, bipartite, no_loop.

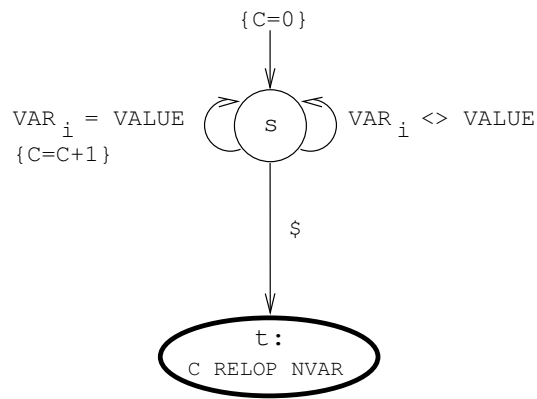


Figure 4.110: Automaton of the count constraint

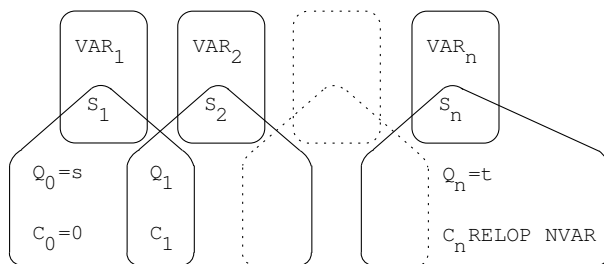


Figure 4.111: Hypergraph of the reformulation corresponding to the automaton of the count constraint

- Remark** Similar to the among constraint.
- See also** among, counts, nvalue, max_nvalue, min_nvalue.
- Key words** value constraint, counting constraint, automaton, automaton with counters, alpha-acyclic constraint network(2).

4.52 counts

Origin	Derived from count.
Constraint	counts(VALUEs, VARIABLEs, RELOP, LIMIT)
Argument(s)	VALUEs : collection(val – int) VARIABLEs : collection(var – dvar) RELOP : atom LIMIT : dvar
Restriction(s)	required(VALUEs, val) distinct(VALUEs, val) required(VARIABLEs, var) RELOP ∈ [=, ≠, <, ≥, >, ≤]
Purpose	Let N be the number of variables of the VARIABLEs collection assigned to a value of the VALUEs collection. Enforce condition N RELOP LIMIT to hold.
Arc input(s)	VARIABLEs VALUEs
Arc generator	PRODUCT \mapsto collection(variables, values)
Arc arity	2
Arc constraint(s)	variables.var = values.val
Graph property(ies)	NARC RELOP LIMIT
Example	$\text{counts} \left(\left(\begin{array}{l} \{\text{val} - 1, \text{val} - 3, \text{val} - 4, \text{val} - 9\}, \\ \left\{ \begin{array}{l} \text{var} - 4, \\ \text{var} - 5, \\ \text{var} - 5, \\ \text{var} - 4, \\ \text{var} - 1, \\ \text{var} - 5 \end{array} \right\} \right), =, 3 \right)$ <p>The constraint holds since values 1, 3, 4 and 9 are used by three variables of the VARIABLEs collection. This number is equal to the last argument of the counts constraint. Parts (A) and (B) of Figure 4.112 respectively show the initial and final graph. Since we use the NARC graph property, the arcs of the final graph are stressed in bold.</p>
Graph model	Because of the arc constraint variables.var = values.val and since each domain variable can take at most one value, NARC is the number of variables taking a value in the VALUEs collection.
Automaton	Figure 4.113 depicts the automaton associated to the counts constraint. To each variable VAR _{<i>i</i>} of the collection VARIABLEs corresponds a 0-1 signature variable S _{<i>i</i>} . The following signature constraint links VAR _{<i>i</i>} and S _{<i>i</i>} : VAR _{<i>i</i>} ∈ VALUEs ⇔ S _{<i>i</i>} .

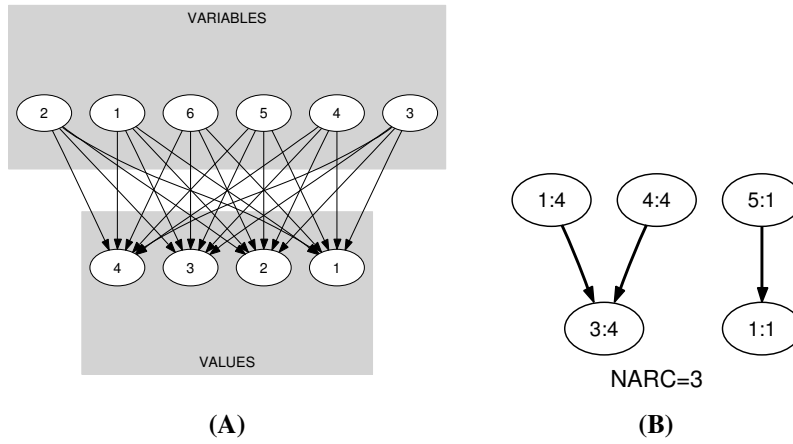


Figure 4.112: Initial and final graph of the counts constraint

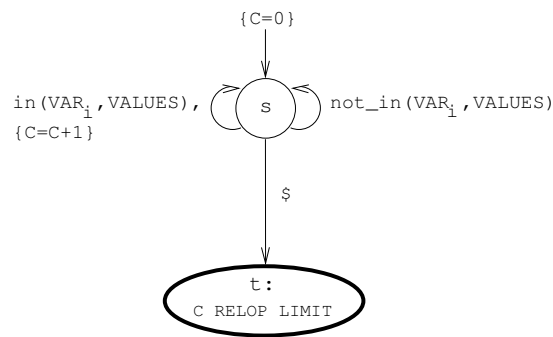


Figure 4.113: Automaton of the counts constraint

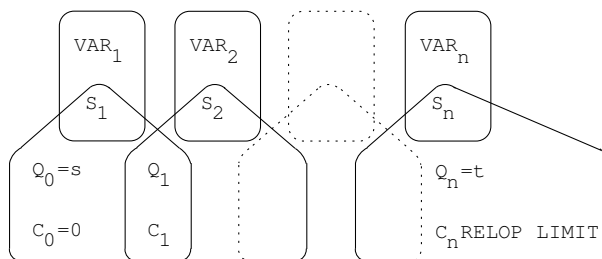


Figure 4.114: Hypergraph of the reformulation corresponding to the automaton of the counts constraint

Usage	Used in the Constraint(s) on sets slot for defining some constraints like <code>assign_and_counts</code> .
Used in	<code>assign_and_counts</code> .
See also	<code>count</code> , <code>among</code> .
Key words	value constraint, counting constraint, automaton, automaton with counters, alpha-acyclic constraint network(2), acyclic, bipartite, <code>no_loop</code> .

20030820

357

4.53 crossing

Origin	Inspired by [84].
Constraint	<code>crossing(NCROSS, SEGMENTS)</code>
Argument(s)	<code>NCROSS</code> : dvar <code>SEGMENTS</code> : <code>collection(ox - dvar, oy - dvar, ex - dvar, ey - dvar)</code>
Restriction(s)	$NCROSS \geq 0$ $NCROSS \leq (SEGMENTS * SEGMENTS - SEGMENTS)/2$ <code>required(SEGMENTS, [ox, oy, ex, ey])</code>
Purpose	<code>NCROSS</code> is the number of line-segments intersections between the line-segments defined by the <code>SEGMENTS</code> collection. Each line-segment is defined by the coordinates (ox, oy) and (ex, ey) of its two extremities.
Arc input(s)	<code>SEGMENTS</code>
Arc generator	$CLIQUE(\langle) \mapsto \text{collection}(s1, s2)$
Arc arity	2
Arc constraint(s)	<ul style="list-style-type: none"> • $\max(s1.ox, s1.ex) \geq \min(s2.ox, s2.ex)$ • $\max(s2.ox, s2.ex) \geq \min(s1.ox, s1.ex)$ • $\max(s1.oy, s1.ey) \geq \min(s2.oy, s2.ey)$ • $\max(s2.oy, s2.ey) \geq \min(s1.oy, s1.ey)$ • $\bigvee \left(\begin{array}{l} (s2.ox - s1.ex) * (s1.ey - s1.oy) - (s1.ex - s1.ox) * (s2.oy - s1.ey) = 0, \\ (s2.ex - s1.ex) * (s2.oy - s1.oy) - (s2.ox - s1.ox) * (s2.ey - s1.ey) = 0, \\ \text{sign}((s2.ox - s1.ex) * (s1.ey - s1.oy) - (s1.ex - s1.ox) * (s2.oy - s1.ey)) \neq \\ \text{sign}((s2.ex - s1.ex) * (s2.oy - s1.oy) - (s2.ox - s1.ox) * (s2.ey - s1.ey)) \end{array} \right)$
Graph property(ies)	$NARC = NCROSS$
Example	$\text{crossing} \left(3, \left\{ \begin{array}{l} ox - 1 \quad oy - 4 \quad ex - 9 \quad ey - 2, \\ ox - 1 \quad oy - 1 \quad ex - 3 \quad ey - 5, \\ ox - 3 \quad oy - 2 \quad ex - 7 \quad ey - 4, \\ ox - 9 \quad oy - 1 \quad ex - 9 \quad ey - 4 \end{array} \right\} \right)$ <p>Parts (A) and (B) of Figure 4.115 respectively show the initial and final graph. Since we use the NARC graph property, the arcs of the final graph are stressed in bold. An arc constraint expresses the fact the two line-segments intersect. It is taken from [84, page 889]. Each arc of the final graph corresponds to a line-segments intersection. Figure 4.116 gives a picture of the previous example, where one can observe three line-segments intersections.</p>
Graph model	Each line-segment is described by the x and y coordinates of its two extremities. In the arc generator we use the restriction \langle in order to generate one single arc for each pair of segments. This is required, since otherwise we would count more than once a given line-segments intersection.

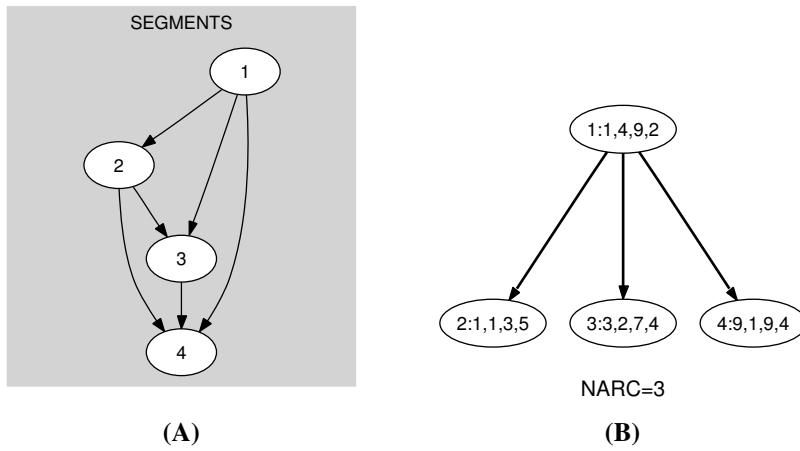


Figure 4.115: Initial and final graph of the crossing constraint

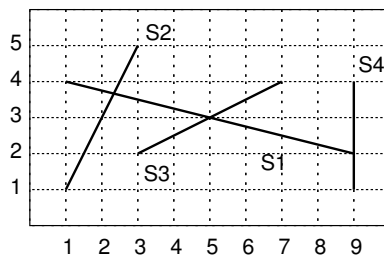


Figure 4.116: Intersection between line-segments

See also `graph_crossing`, `two_layer_edge_crossing`.

Key words geometrical constraint, line-segments intersection, no_loop.

20000128

361

4.54 cumulative

Origin	[67]
Constraint	<code>cumulative(TASKS, LIMIT)</code>
Argument(s)	<code>TASKS : collection(origin - dvar, duration - dvar, end - dvar, height - dvar)</code> <code>LIMIT : int</code>
Restriction(s)	<code>require_at_least(2, TASKS, [origin, duration, end])</code> <code>required(TASKS, height)</code> <code>TASKS.duration ≥ 0</code> <code>TASKS.height ≥ 0</code> <code>LIMIT ≥ 0</code>
Purpose	Cumulative scheduling constraint or scheduling under resource constraints. Consider a set \mathcal{T} of tasks described by the <code>TASKS</code> collection. The <code>cumulative</code> constraint enforces that at each point in time, the cumulated height of the set of tasks that overlap that point, does not exceed a given limit. It also imposes for each task of \mathcal{T} the constraint <code>origin + duration = end</code> .
Arc input(s)	<code>TASKS</code>
Arc generator	<code>SELF</code> \mapsto <code>collection(tasks)</code>
Arc arity	1
Arc constraint(s)	<code>tasks.origin + tasks.duration = tasks.end</code>
Graph property(ies)	$\overline{\text{NARC}} = \text{TASKS} $
Arc input(s)	<code>TASKS TASKS</code>
Arc generator	<code>PRODUCT</code> \mapsto <code>collection(tasks1, tasks2)</code>
Arc arity	2
Arc constraint(s)	<ul style="list-style-type: none"> • <code>tasks1.duration > 0</code> • <code>tasks2.origin ≤ tasks1.origin</code> • <code>tasks1.origin < tasks2.end</code>
Sets	$\text{SUCC} \mapsto \left[\begin{array}{l} \text{source,} \\ \text{variables} - \text{col} \left(\begin{array}{l} \text{VARIABLES} - \text{collection}(\text{var} - \text{dvar}), \\ [\text{item}(\text{var} - \text{TASKS.height})] \end{array} \right) \end{array} \right]$
Constraint(s) on sets	<code>sum_ctr(variables, ≤, LIMIT)</code>

Example

$$\text{cumulative} \left(\left\{ \begin{array}{l} \text{origin} - 1 \quad \text{duration} - 3 \quad \text{end} - 4 \quad \text{height} - 1, \\ \text{origin} - 2 \quad \text{duration} - 9 \quad \text{end} - 11 \quad \text{height} - 2, \\ \text{origin} - 3 \quad \text{duration} - 10 \quad \text{end} - 13 \quad \text{height} - 1, \\ \text{origin} - 6 \quad \text{duration} - 6 \quad \text{end} - 12 \quad \text{height} - 1, \\ \text{origin} - 7 \quad \text{duration} - 2 \quad \text{end} - 9 \quad \text{height} - 3 \end{array} \right\}, 8 \right)$$

Parts (A) and (B) of Figure 4.117 respectively show the initial and final graph associated to the second graph constraint. On the one hand, each source vertex of the final graph can be interpreted as a time point. On the other hand the successors of a source vertex correspond to those tasks which overlap that time point. The cumulative constraint holds since for each successor set S of the final graph the sum of the heights of the tasks in S does not exceed the limit $\text{LIMIT} = 8$. Figure 4.118 shows the cumulated profile associated to the previous example.

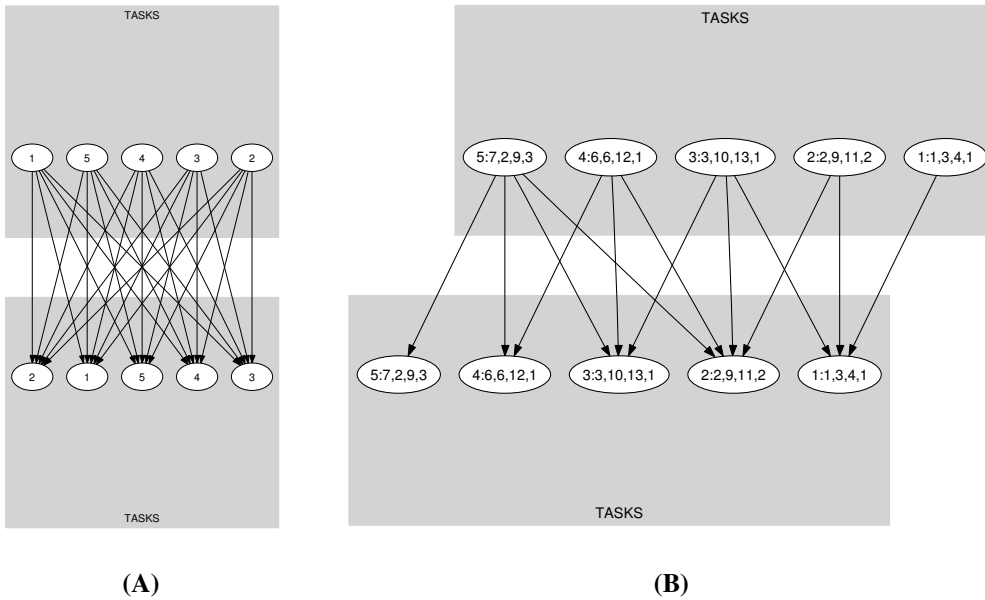


Figure 4.117: Initial and final graph of the cumulative constraint

Graph model

The first graph constraint enforces for each task the link between its origin, its duration and its end. The second graph constraint makes sure, for each time point t corresponding to the start of a task, that the cumulated heights of the tasks that overlap t does not exceed the limit of the resource.

Signature

Since TASKS is the maximum number of vertices of the final graph of the first graph constraint we can rewrite $\text{NARC} = |\text{TASKS}|$ to $\text{NARC} \geq |\text{TASKS}|$. This leads to simplify $\underline{\text{NARC}}$ to $\overline{\text{NARC}}$.

Automaton

Figure 4.119 depicts the automaton associated to the cumulative constraint. To each item of the collection TASKS corresponds a signature variable S_i , which is equal to 1.

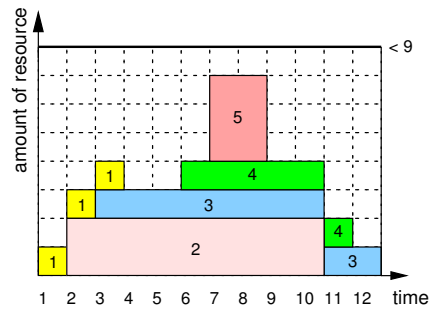


Figure 4.118: Resource consumption profile

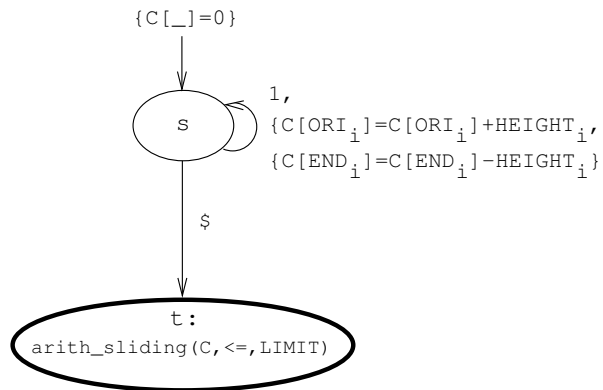


Figure 4.119: Automaton of the cumulative constraint

- Algorithm** [85, 86, 87]. Within the context of linear programming, the reference [8] provides a relaxation of the `cumulative` constraint.
- See also** `bin_packing`, `cumulative_product`, `coloured_cumulative`, `cumulative_two_d`, `coloured_cumulatives`, `cumulatives`, `cumulative_with_level_of_priority`.
- Key words** scheduling constraint, resource constraint, temporal constraint, linear programming, producer-consumer, squared squares, automaton, automaton with array of counters.

4.55 cumulative_product

Origin	Derived from cumulative.
Constraint	cumulative_product(TASKS, LIMIT)
Argument(s)	TASKS : collection(origin - dvar, duration - dvar, end - dvar, height - dvar) LIMIT : int
Restriction(s)	require_at_least(2, TASKS, [origin, duration, end]) required(TASKS, height) TASKS.duration \geq 0 TASKS.height \geq 1 LIMIT \geq 0
Purpose	Consider a set \mathcal{T} of tasks described by the TASKS collection. The cumulative_product constraint enforces that at each point in time, the product of the height of the set of tasks that overlap that point, does not exceed a given limit. It also imposes for each task of \mathcal{T} the constraint origin + duration = end.
<hr/>	
Arc input(s)	TASKS
Arc generator	<i>SELF</i> \mapsto collection(tasks)
Arc arity	1
Arc constraint(s)	tasks.origin + tasks.duration = tasks.end
Graph property(ies)	$\overline{\text{NARC}} = \text{TASKS} $
<hr/>	
Arc input(s)	TASKS TASKS
Arc generator	<i>PRODUCT</i> \mapsto collection(tasks1, tasks2)
Arc arity	2
Arc constraint(s)	<ul style="list-style-type: none"> • tasks1.duration > 0 • tasks2.origin \leq tasks1.origin • tasks1.origin < tasks2.end
Sets	SUCC \mapsto $\left[\begin{array}{l} \text{source,} \\ \text{variables - col} \left(\begin{array}{l} \text{VARIABLES - collection(var - dvar),} \\ \text{[item(var - ITEMS.height)]} \end{array} \right) \end{array} \right]$
Constraint(s) on sets	product_ctr(variables, \leq , LIMIT)

Example

$$\text{cumulative_product} \left(\left(\begin{array}{cccc} \text{origin} - 1 & \text{duration} - 3 & \text{end} - 4 & \text{height} - 1, \\ \text{origin} - 2 & \text{duration} - 9 & \text{end} - 11 & \text{height} - 2, \\ \text{origin} - 3 & \text{duration} - 10 & \text{end} - 13 & \text{height} - 1, \\ \text{origin} - 6 & \text{duration} - 6 & \text{end} - 12 & \text{height} - 1, \\ \text{origin} - 7 & \text{duration} - 2 & \text{end} - 9 & \text{height} - 3 \end{array} \right), 6 \right)$$

Parts (A) and (B) of Figure 4.120 respectively show the initial and final graph associated to the second graph constraint. On the one hand, each source vertex of the final graph can be interpreted as a time point. On the other hand the successors of a source vertex correspond to those tasks which overlap that time point. The `cumulative_product` constraint holds since for each successor set \mathcal{S} of the final graph the product of the heights of the tasks in \mathcal{S} does not exceed the limit $\text{LIMIT} = 6$. Figure 4.121 shows the solution associated to the previous example.

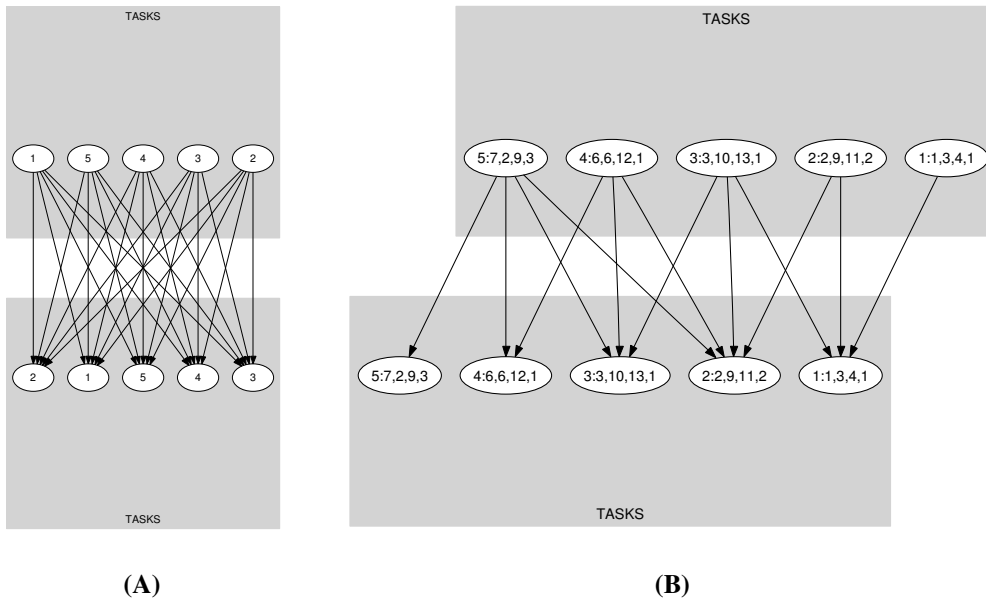


Figure 4.120: Initial and final graph of the `cumulative_product` constraint

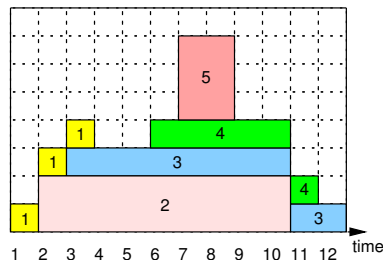


Figure 4.121: Solution of the `cumulative_product` constraint

- Signature** Since *TASKS* is the maximum number of vertices of the final graph of the first graph constraint we can rewrite $\text{NARC} = |\text{TASKS}|$ to $\text{NARC} \geq |\text{TASKS}|$. This leads to simplify $\overline{\text{NARC}}$ to $\overline{\text{NARC}}$.
- See also** *cumulative*.
- Key words** scheduling constraint, resource constraint, temporal constraint, product.

4.56 cumulative_two_d

Origin Inspired by cumulative and diffn.

Constraint cumulative_two_d(RECTANGLES, LIMIT)

Argument(s)

RECTANGLES	:	collection	(start1 - dvar, size1 - dvar, last1 - dvar, start2 - dvar, size2 - dvar, last2 - dvar, height - dvar)
LIMIT	:	int			

Restriction(s)

```

require_at_least(2, RECTANGLES, [start1, size1, last1])
require_at_least(2, RECTANGLES, [start2, size2, last2])
required(RECTANGLES, height)
RECTANGLES.size1 ≥ 0
RECTANGLES.size2 ≥ 0
RECTANGLES.height ≥ 0
LIMIT ≥ 0
  
```

Purpose

Consider a set \mathcal{R} of rectangles described by the RECTANGLES collection. Enforces that at each point of the plane, the cumulated height of the set of rectangles that overlap that point, does not exceed a given limit.

Derived Collection(s)

col	CORNERS	-	collection	(size1 - dvar, size2 - dvar, x - dvar, y - dvar,)
	item	(size1 - RECTANGLES.size1, size2 - RECTANGLES.size2, x - RECTANGLES.start1, y - RECTANGLES.start2)	,	
	item	(size1 - RECTANGLES.size1, size2 - RECTANGLES.size2, x - RECTANGLES.start1, y - RECTANGLES.last2)	,	
	item	(size1 - RECTANGLES.size1, size2 - RECTANGLES.size2, x - RECTANGLES.last1, y - RECTANGLES.start2)	,	
	item	(size1 - RECTANGLES.size1, size2 - RECTANGLES.size2, x - RECTANGLES.last1, y - RECTANGLES.last2)	,	

Arc input(s) RECTANGLES

Arc generator *SELF* \mapsto collection(rectangles)

Arc arity	1
Arc constraint(s)	<ul style="list-style-type: none"> • <code>rectangles.start1 + rectangles.size1 - 1 = rectangles.last1</code> • <code>rectangles.start2 + rectangles.size2 - 1 = rectangles.last2</code>
Graph property(ies)	$\text{NARC} = \text{RECTANGLES} $
Arc input(s)	CORNERS RECTANGLES
Arc generator	$\text{PRODUCT} \mapsto \text{collection}(\text{corners}, \text{rectangles})$
Arc arity	2
Arc constraint(s)	<ul style="list-style-type: none"> • <code>corners.size1 > 0</code> • <code>corners.size2 > 0</code> • <code>rectangles.start1 ≤ corners.x</code> • <code>corners.x ≤ rectangles.last1</code> • <code>rectangles.start2 ≤ corners.y</code> • <code>corners.y ≤ rectangles.last2</code>
Sets	$\text{SUCC} \mapsto \left[\begin{array}{l} \text{source,} \\ \text{variables} - \text{col} \left(\begin{array}{l} \text{VARIABLES} - \text{collection}(\text{var} - \text{dvar}), \\ [\text{item}(\text{var} - \text{RECTANGLES.height})] \end{array} \right) \end{array} \right]$
Constraint(s) on sets	$\text{sum_ctr}(\text{variables}, \leq, \text{LIMIT})$
Example	$\text{cumulative_two_d} \left(\left\{ \begin{array}{l} \text{start1} - 1 \quad \text{size1} - 4 \quad \text{last1} - 4 \quad \text{start2} - 3 \quad \text{size2} - 3 \quad \text{last2} - 5 \quad \text{height} - 4, \\ \text{start1} - 3 \quad \text{size1} - 2 \quad \text{last1} - 4 \quad \text{start2} - 1 \quad \text{size2} - 2 \quad \text{last2} - 2 \quad \text{height} - 2, \\ \text{start1} - 1 \quad \text{size1} - 2 \quad \text{last1} - 2 \quad \text{start2} - 1 \quad \text{size2} - 2 \quad \text{last2} - 2 \quad \text{height} - 3, \\ \text{start1} - 4 \quad \text{size1} - 1 \quad \text{last1} - 4 \quad \text{start2} - 1 \quad \text{size2} - 1 \quad \text{last2} - 1 \quad \text{height} - 1 \end{array} \right\}, 4 \right)$ <p>Parts (A) and (B) of Figure 4.122 respectively show the initial and final graph associated to the second graph constraint. On the one hand, each source vertex of the final graph corresponds to the corner of a rectangle of the RECTANGLES collection. On the other hand the successors of a source vertex are those rectangles which overlap that corner.</p> <p>Part (A) of Figure 4.123 shows 4 rectangles of height 4, 2, 3 and 1. Part (B) gives the corresponding cumulated 2-dimensional profile, where each number is the cumulated height of all the rectangles that contain the corresponding region.</p>
Signature	Since RECTANGLES is the maximum number of vertices of the final graph of the first graph constraint we can rewrite $\text{NARC} = \text{RECTANGLES} $ to $\text{NARC} \geq \text{RECTANGLES} $. This leads to simplify $\overline{\text{NARC}}$ to $\overline{\text{NARC}}$.
Usage	The <code>cumulative_two_d</code> constraint is a necessary condition for the <code>diffn</code> constraint in 3 dimensions (i.e. the placement of parallelepipeds in such a way that they do not pairwise overlap and that each parallelepiped has his sides parallel to the sides of the placement space).
Algorithm	A first natural way to handle this constraint would be to accumulate the <i>compulsory parts</i> [85] of the rectangles in a quadtree [88]. To each leaf of the quadtree we associate the cumulated height of the rectangles containing the corresponding region.

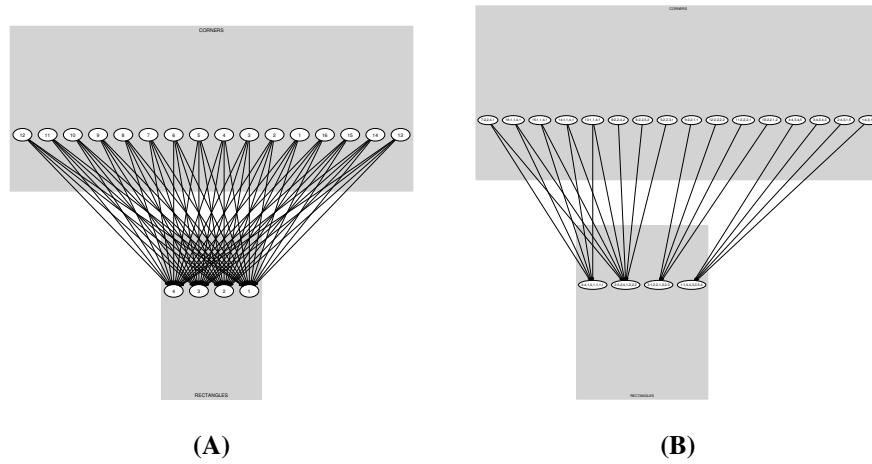


Figure 4.122: Initial and final graph of the `cumulative_two_d` constraint

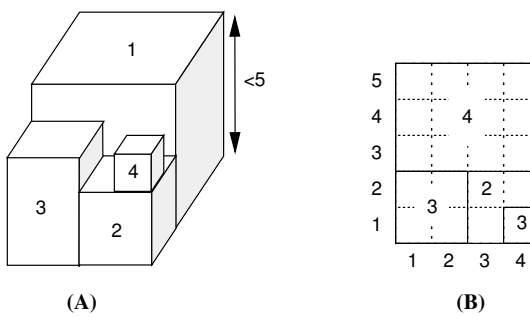


Figure 4.123: Two representations of a 2-dimensional cumulated profile

20000128

373

See also cumulative, diffn, bin_packing.

Key words geometrical constraint, derived collection.

4.57 cumulative_with_level_of_priority

Origin	H. Simonis
Constraint	cumulative_with_level_of_priority(TASKS, PRIORITIES)
Argument(s)	$\begin{array}{l} \text{TASKS} \quad : \text{collection} \left(\begin{array}{l} \text{priority} - \text{int}, \\ \text{origin} - \text{dvar}, \\ \text{duration} - \text{dvar}, \\ \text{end} - \text{dvar}, \\ \text{height} - \text{dvar} \end{array} \right) \\ \text{PRIORITIES} : \text{collection}(\text{id} - \text{int}, \text{capacity} - \text{int}) \end{array}$
Restriction(s)	<pre> required(TASKS, [priority, height]) require_at_least(2, TASKS, [origin, duration, end]) TASKS.priority ≥ 1 TASKS.priority ≤ PRIORITIES TASKS.duration ≥ 0 TASKS.height ≥ 0 required(PRIORITIES, [id, capacity]) PRIORITIES.id ≥ 1 PRIORITIES.id ≤ PRIORITIES increasing_seq(PRIORITIES, id) increasing_seq(PRIORITIES, capacity) </pre>

Purpose

Consider a set \mathcal{T} of tasks described by the TASKS collection where each task has a given priority chosen in the range $[1, \text{PRIORITIES}]$. Let \mathcal{T}_i denotes the subset of tasks of \mathcal{T} which all have a priority less than or equal to i . For each set \mathcal{T}_i , the cumulative_with_level_of_priority constraint enforces that at each point in time, the cumulated height of the set of tasks that overlap that point, does not exceed a given limit. Finally, it also imposes for each task of \mathcal{T} the constraint $\text{origin} + \text{duration} = \text{end}$.

Derived Collection(s)	$\text{col} \left(\begin{array}{l} \text{TIME_POINTS} - \text{collection}(\text{idp} - \text{int}, \text{duration} - \text{dvar}, \text{point} - \text{dvar}), \\ \left[\begin{array}{l} \text{item}(\text{idp} - \text{TASKS.priority}, \text{duration} - \text{TASKS.duration}, \text{point} - \text{TASKS.origin}), \\ \text{item}(\text{idp} - \text{TASKS.priority}, \text{duration} - \text{TASKS.duration}, \text{point} - \text{TASKS.end}) \end{array} \right] \end{array} \right)$
Arc input(s)	TASKS
Arc generator	$\text{SELF} \mapsto \text{collection}(\text{tasks})$
Arc arity	1
Arc constraint(s)	$\text{tasks.origin} + \text{tasks.duration} = \text{tasks.end}$
Graph property(ies)	$\text{NARC} = \text{TASKS} $

For all items of PRIORITIES:

Arc input(s)	TIME_POINTS TASKS
Arc generator	$PRODUCT \mapsto \text{collection}(\text{time_points}, \text{tasks})$
Arc arity	2
Arc constraint(s)	<ul style="list-style-type: none"> • $\text{time_points.idp} = \text{PRIORITIES.id}$ • $\text{time_points.idp} \geq \text{tasks.priority}$ • $\text{time_points.duration} > 0$ • $\text{tasks.origin} \leq \text{time_points.point}$ • $\text{time_points.point} < \text{tasks.end}$
Sets	$\text{SUCC} \mapsto \left[\begin{array}{l} \text{source,} \\ \text{variables} - \text{col} \left(\begin{array}{l} \text{VARIABLES} - \text{collection}(\text{var} - \text{dvar}), \\ [\text{item}(\text{var} - \text{TASKS.height})] \end{array} \right) \end{array} \right]$
Constraint(s) on sets	$\text{sum_ctr}(\text{variables}, \leq, \text{PRIORITIES.capacity})$

Example	$\text{cumulative_with_level_of_priority} \left(\left\{ \begin{array}{l} \left\{ \begin{array}{l} \text{priority} - 1 \quad \text{origin} - 1 \quad \text{duration} - 2 \quad \text{end} - 3 \quad \text{height} - 1, \\ \text{priority} - 1 \quad \text{origin} - 2 \quad \text{duration} - 3 \quad \text{end} - 5 \quad \text{height} - 1, \\ \text{priority} - 1 \quad \text{origin} - 5 \quad \text{duration} - 2 \quad \text{end} - 7 \quad \text{height} - 2, \\ \text{priority} - 2 \quad \text{origin} - 3 \quad \text{duration} - 2 \quad \text{end} - 5 \quad \text{height} - 2, \\ \text{priority} - 2 \quad \text{origin} - 6 \quad \text{duration} - 3 \quad \text{end} - 9 \quad \text{height} - 1 \end{array} \right\}, \\ \left\{ \begin{array}{l} \text{id} - 1 \quad \text{capacity} - 2, \\ \text{id} - 2 \quad \text{capacity} - 3 \end{array} \right\} \end{array} \right)$
----------------	---

Within the context of the second graph constraint, part (A) of Figure 4.124 shows the initial graphs associated to priorities 1 and 2. Part (B) of Figure 4.124 shows the corresponding final graphs associated to priorities 1 and 2. On the one hand, each source vertex of the final graph can be interpreted as a time point p . On the other hand the successors of a source vertex correspond to those tasks which both overlap that time point p and have a priority less than or equal to a given level. The `cumulative_with_level_of_priority` constraint holds since for each successor set \mathcal{S} of the final graph the sum of the height of the tasks in \mathcal{S} is less than or equal to the capacity associated to a given level of priority. Figure 4.125 shows the cumulated profile associated to both levels of priority.

Signature Since TASKS is the maximum number of vertices of the final graph of the first graph constraint we can rewrite $\text{NARC} = |\text{TASKS}|$ to $\text{NARC} \geq |\text{TASKS}|$. This leads to simplify NARC to $\overline{\text{NARC}}$.

Usage The `cumulative_with_level_of_priority` constraint was suggested by problems from the telecommunication area where one has to ensure different levels of quality of service. For this purpose the capacity of a transmission link is splitted so that a given percentage is reserved to each level. In addition we have that, if the capacities allocated to levels $1, 2, \dots, i$ is not completely used, then level $i+1$ can use the corresponding spare capacity.

Remark The `cumulative_with_level_of_priority` constraint can be modeled by a conjunction of cumulative constraints. As shown by the next example, the consistency for all variables of the cumulative constraints does not implies consistency for the corresponding `cumulative_with_level_of_priority` constraint. The following `cumulative_with_level_of_priority` constraint

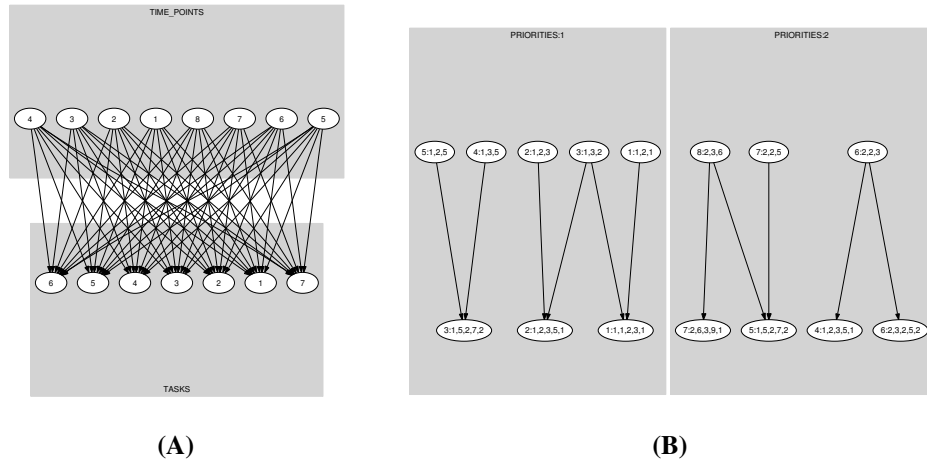


Figure 4.124: Initial and final graph of the `cumulative_with_level_of_priority` constraint

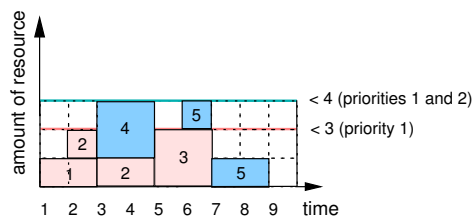


Figure 4.125: Resource consumption profile according to both levels of priority

$$\text{cumulative_with_level_of_priority} \left(\left(\begin{array}{l} \left\{ \begin{array}{l} \text{priority} - 1 \quad \text{origin} - o_1 \quad \text{duration} - 2 \quad \text{height} - 2, \\ \text{priority} - 1 \quad \text{origin} - o_2 \quad \text{duration} - 2 \quad \text{height} - 1, \\ \text{priority} - 2 \quad \text{origin} - o_3 \quad \text{duration} - 1 \quad \text{height} - 3 \end{array} \right\}, \\ \left\{ \begin{array}{l} \text{id} - 1 \quad \text{capacity} - 2, \\ \text{id} - 2 \quad \text{capacity} - 3 \end{array} \right\} \end{array} \right) \right)$$

where the domains of o_1 , o_2 and o_3 are respectively equal to $\{1, 2, 3\}$, $\{1, 2, 3\}$ and $\{1, 2, 3, 4\}$ corresponds to the following conjunction of cumulative constraints

$$\text{cumulative} \left(\left\{ \begin{array}{l} \text{origin} - o_1 \quad \text{duration} - 2 \quad \text{height} - 2, \\ \text{origin} - o_2 \quad \text{duration} - 2 \quad \text{height} - 1 \end{array} \right\}, 2 \right)$$

$$\text{cumulative} \left(\left\{ \begin{array}{l} \text{origin} - o_1 \quad \text{duration} - 2 \quad \text{height} - 2, \\ \text{origin} - o_2 \quad \text{duration} - 2 \quad \text{height} - 1, \\ \text{origin} - o_3 \quad \text{duration} - 1 \quad \text{height} - 3 \end{array} \right\}, 3 \right)$$

Even if the cumulative could achieve arc-consistency, the previous conjunction of cumulative constraints would not detect the fact that there is no solution.

See also cumulative.

Key words scheduling constraint, resource constraint, temporal constraint, derived collection.

4.58 cumulatives

Origin	[89]
Constraint	<code>cumulatives(TASKS, MACHINES, CTR)</code>
Argument(s)	$\begin{array}{l} \text{TASKS} \quad : \quad \text{collection} \left(\begin{array}{l} \text{machine} - \text{dvar}, \\ \text{origin} - \text{dvar}, \\ \text{duration} - \text{dvar}, \\ \text{end} - \text{dvar}, \\ \text{height} - \text{dvar} \end{array} \right) \\ \text{MACHINES} \quad : \quad \text{collection}(\text{id} - \text{int}, \text{capacity} - \text{int}) \\ \text{CTR} \quad \quad : \quad \text{atom} \end{array}$
Restriction(s)	<pre>required(TASKS, [machine, height]) require_at_least(2, TASKS, [origin, duration, end]) in_attr(TASKS, machine, MACHINES, id) TASKS.duration ≥ 0 required(MACHINES, [id, capacity]) distinct(MACHINES, id) CTR ∈ [≤, ≥]</pre>
Purpose	<p>Consider a set \mathcal{T} of tasks described by the <code>TASKS</code> collection. When <code>CTR</code> is equal to \leq (repectively \geq), the <code>cumulatives</code> constraint enforces the following condition for each machine m: At each point in time, where at least one task assigned on machine m is present, the cumulated height of the set of tasks that both overlap that point and are assigned to machine m should be less than or equal to (repectively greater than or equal to) the capacity associated to machine m. It also imposes for each task of \mathcal{T} the constraint <code>origin + duration = end</code>.</p>
Derived Collection(s)	$\text{col} \left(\begin{array}{l} \text{TIME_POINTS} - \text{collection}(\text{idm} - \text{int}, \text{duration} - \text{dvar}, \text{point} - \text{dvar}), \\ \left[\begin{array}{l} \text{item}(\text{idm} - \text{TASKS.machine}, \text{duration} - \text{TASKS.duration}, \text{point} - \text{TASKS.origin}), \\ \text{item}(\text{idm} - \text{TASKS.machine}, \text{duration} - \text{TASKS.duration}, \text{point} - \text{TASKS.end}) \end{array} \right] \end{array} \right)$
Arc input(s)	<code>TASKS</code>
Arc generator	<code>SELF</code> \mapsto <code>collection(tasks)</code>
Arc arity	1
Arc constraint(s)	<code>tasks.origin + tasks.duration = tasks.end</code>
Graph property(ies)	<code>NARC = TASKS </code>
	<p>For all items of <code>MACHINES</code>:</p>
Arc input(s)	<code>TIME_POINTS TASKS</code>
Arc generator	<code>PRODUCT</code> \mapsto <code>collection(time_points, tasks)</code>

Arc arity 2

Arc constraint(s)

- `time_points.idm = MACHINES.id`
- `time_points.idm = tasks.machine`
- `time_points.duration > 0`
- `tasks.origin ≤ time_points.point`
- `time_points.point < tasks.end`

Sets

$$\text{SUCC} \mapsto \left[\begin{array}{l} \text{source,} \\ \text{variables} - \text{col} \left(\begin{array}{l} \text{VARIABLES} - \text{collection}(\text{var} - \text{dvar}), \\ [\text{item}(\text{var} - \text{TASKS.height})] \end{array} \right) \end{array} \right]$$

Constraint(s) on sets `sum_ctr(variables, CTR, MACHINES.capacity)`

Example

$$\text{cumulatives} \left(\left\{ \begin{array}{l} \text{machine} - 1 \quad \text{origin} - 2 \quad \text{duration} - 2 \quad \text{end} - 4 \quad \text{height} - -2, \\ \text{machine} - 1 \quad \text{origin} - 1 \quad \text{duration} - 4 \quad \text{end} - 5 \quad \text{height} - -1, \\ \text{machine} - 1 \quad \text{origin} - 4 \quad \text{duration} - 2 \quad \text{end} - 6 \quad \text{height} - -1, \\ \text{machine} - 1 \quad \text{origin} - 2 \quad \text{duration} - 3 \quad \text{end} - 5 \quad \text{height} - -2, \\ \text{machine} - 1 \quad \text{origin} - 5 \quad \text{duration} - 2 \quad \text{end} - 7 \quad \text{height} - -2, \\ \text{machine} - 2 \quad \text{origin} - 3 \quad \text{duration} - 2 \quad \text{end} - 5 \quad \text{height} - -1, \\ \text{machine} - 2 \quad \text{origin} - 1 \quad \text{duration} - 4 \quad \text{end} - 5 \quad \text{height} - -1 \end{array} \right\}, \right)$$

`{id - 1 capacity - 0, id - 2 capacity - 0}, ≥`

Within the context of the second graph constraint, part (A) of Figure 4.126 shows the initial graphs associated to machines 1 and 2. Part (B) of Figure 4.126 shows the corresponding final graphs associated to machines 1 and 2. On the one hand, each source vertex of the final graph can be interpreted as a time point p on a specific machine m . On the other hand the successors of a source vertex correspond to those tasks which both overlap that time point p and are assigned to machine m . Since they don't have any successors we have eliminated those vertices corresponding to the end of the last three tasks of the TASKS collection. The `cumulatives` constraint holds since for each successor set \mathcal{S} of the final graph the sum of the height of the tasks in \mathcal{S} is greater than or equal to the capacity of the machine corresponding to the time point associated to \mathcal{S} . Figure 4.127 shows with a thick line the cumulated profile on both machines.

Signature Since TASKS is the maximum number of vertices of the final graph of the first graph constraint we can rewrite `NARC = |TASKS|` to `NARC ≥ |TASKS|`. This leads to simplify NARC to NARC.

Usage As shown in the previous example, the `cumulatives` constraint is useful for covering problems where different demand profiles have to be covered by a set of tasks. This is modelled in the following way:

- To each demand profile is associated a given machine m and a set of tasks for which all attributes (`machine`, `origin`, `duration`, `end`, `height`) are fixed; moreover the `machine` attribute is fixed to m and the `height` attribute is strictly negative. For each machine m the cumulated profile of all the previous tasks constitutes the demand profile to cover.
- To each task that can be used to cover the demand is associated a task for which the `height` attribute is a positive integer; the `height` attribute describes the amount of

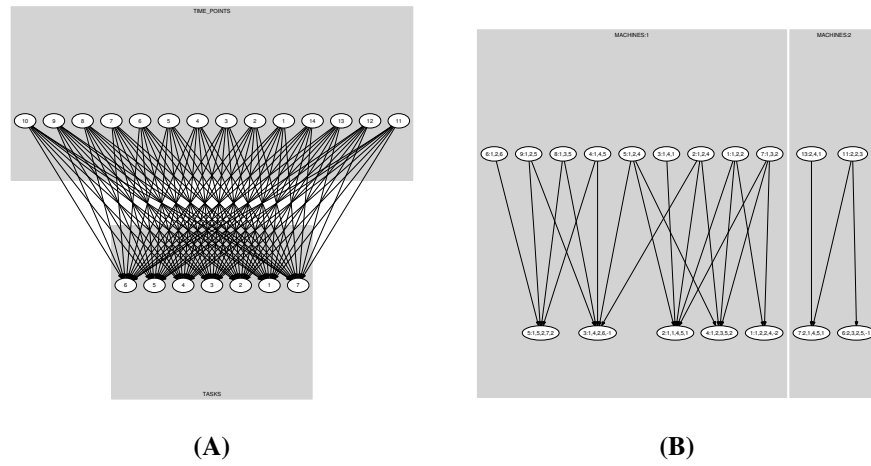


Figure 4.126: Initial and final graph of the cumulatives constraint

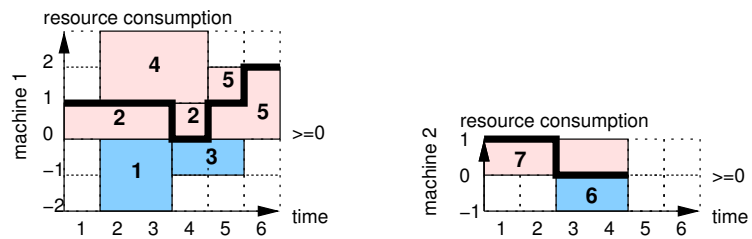


Figure 4.127: Resource consumption profile on the different machines

demand that can be covered by the task at each instant during its execution (between its `origin` and its `end`) on the demand profile associated to the `machine` attribute.

- In order to express the fact that each demand profile should completely be covered, we set the `capacity` attribute of each machine to 0. We can also relax the constraint by setting the `capacity` attribute to a negative number that specifies the maximum allowed uncovered demand at each instant.

The demand profiles might also not be completely fixed in advance.

When all the heights of the tasks are non-negative, one other possible use of the `cumulatives` constraint is to enforce to reach a minimum level of resource consumption. This is imposed on those time-points that are overlapped by at least one task.

By introducing a dummy task of height 0, of origin the minimum origin of all the tasks and of end the maximum end of all the tasks, this can also be imposed between the first and the last utilisation of the resource.

Finally the `cumulatives` constraint is also useful for scheduling problems where several *cumulative* machines are available and where you have to assign each task on a specific machine.

Algorithm Three filtering algorithms for this constraint are described in [89].

See also `cumulative`.

Key words scheduling constraint, resource constraint, temporal constraint, producer-consumer, workload covering, demand profile, derived collection.

4.59 cutset

Origin	[90]
Constraint	cutset(SIZE_CUTSET, NODES)
Argument(s)	SIZE_CUTSET : dvar NODES : collection(index – int, succ – sint, bool – dvar)
Restriction(s)	SIZE_CUTSET ≥ 0 SIZE_CUTSET ≤ NODES required(NODES, [index, succ, bool]) NODES.index ≥ 1 NODES.index ≤ NODES distinct(NODES, index) NODES.bool ≥ 0 NODES.bool ≤ 1
Purpose	Consider a digraph G with n vertices described by the NODES collection. Enforces that the subset of kept vertices of cardinality $n - \text{SIZE_CUTSET}$ and their corresponding arcs form a graph without circuit.
<hr/>	
Arc input(s)	NODES
Arc generator	<i>CLIQUE</i> \mapsto collection(nodes1, nodes2)
Arc arity	2
Arc constraint(s)	<ul style="list-style-type: none"> • in_set(nodes2.index, nodes1.succ) • nodes1.bool = 1 • nodes2.bool = 1
Graph property(ies)	<ul style="list-style-type: none"> • MAX_NSCC ≤ 1 • NVERTEX = NODES – SIZE_CUTSET
<hr/>	
Example	$\text{cutset} \left(1, \left\{ \begin{array}{l} \text{index} - 1 \quad \text{succ} - \{2, 3, 4\} \quad \text{bool} - 1, \\ \text{index} - 2 \quad \text{succ} - \{3\} \quad \text{bool} - 1, \\ \text{index} - 3 \quad \text{succ} - \{4\} \quad \text{bool} - 1, \\ \text{index} - 4 \quad \text{succ} - \{1\} \quad \text{bool} - 0 \end{array} \right\} \right)$

Part (A) of Figure 4.128 shows the initial graph from which we have choose to start. It is derived from the set associated to each vertex. Each set describes the potential values of the succ attribute of a given vertex. Part (B) of Figure 4.128 gives the final graph associated to the example. Since we use the **NVERTEX** graph property, the vertices of the final graph are stressed in bold. The **cutset** constraint holds since the final graph does not contain any circuit and since the number of removed vertices **SIZE_CUTSET** is equal to 1.

- Graph model** We use a set of integers for representing the successors of each vertex. Because of the arc constraint, all arcs such that the `bool` attribute of one extremity is equal to 0 are eliminated; Therefore all vertices for which the `bool` attribute is equal to 0 are also eliminated (since they will correspond to isolated vertices). The graph property $\text{MAX_NSCC} \leq 1$ enforces the size of the largest strongly connected component to not exceed 1; Therefore, the final graph can't contain any circuit.
- Usage** The paper [90] introducing the `cutset` constraint mentions applications from various areas such that *deadlock breaking* or *program verification*.
- Algorithm** The filtering algorithm presented in [90] uses graph reduction techniques inspired from Levy and Low [91] as well as from Lloyd, Soffa and Wang [92].
- Key words** graph constraint, circuit, directed acyclic graph.

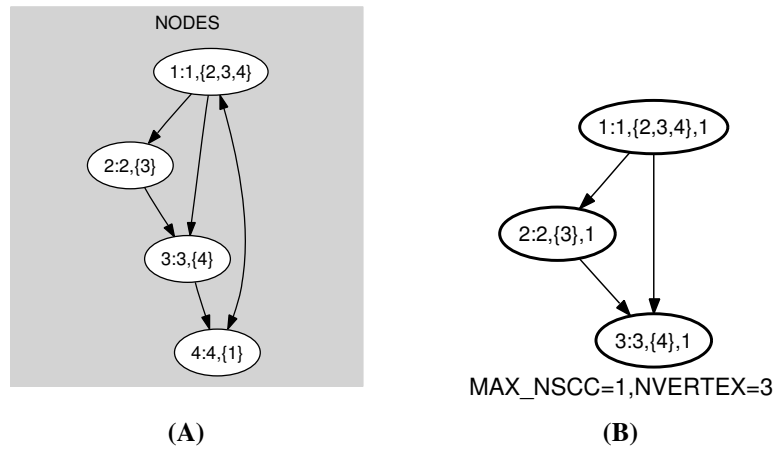


Figure 4.128: Initial and final graph of the cutset set constraint

4.60 cycle

Origin	[37]
Constraint	<code>cycle(NCYCLE, NODES)</code>
Argument(s)	NCYCLE : dvar NODES : <code>collection(index - int, succ - dvar)</code>
Restriction(s)	$NCYCLE \geq 1$ $NCYCLE \leq NODES $ <code>required(NODES, [index, succ])</code> $NODES.index \geq 1$ $NODES.index \leq NODES $ <code>distinct(NODES, index)</code> $NODES.succ \geq 1$ $NODES.succ \leq NODES $

Purpose

Consider a digraph G described by the `NODES` collection. `NCYCLE` is equal to the number of circuits for covering G in such a way that each vertex of G belongs to one single circuit. `NCYCLE` can also be interpreted as the number of cycles of the permutation associated to the successor variables of the `NODES` collection.

Arc input(s)	<code>NODES</code>
Arc generator	<code>CLIQUE</code> \mapsto <code>collection(nodes1, nodes2)</code>
Arc arity	2
Arc constraint(s)	<code>nodes1.succ = nodes2.index</code>
Graph property(ies)	<ul style="list-style-type: none"> • <code>NTREE</code> = 0 • <code>NCC</code> = <code>NCYCLE</code>

Example

$$\text{cycle} \left(2, \left\{ \begin{array}{l} \text{index} - 1 \quad \text{succ} - 2, \\ \text{index} - 2 \quad \text{succ} - 1, \\ \text{index} - 3 \quad \text{succ} - 5, \\ \text{index} - 4 \quad \text{succ} - 3, \\ \text{index} - 5 \quad \text{succ} - 4 \end{array} \right\} \right)$$

In this previous example we have the following two cycles: $1 \rightarrow 2 \rightarrow 1$ and $3 \rightarrow 5 \rightarrow 4 \rightarrow 3$. Parts (A) and (B) of Figure 4.129 respectively show the initial and final graph. Since we use the `NCC` graph property, we show the two connected components of the final graph. The constraint holds since all the vertices belong to a circuit (i.e. `NTREE` = 0) and since `NCYCLE` = `NCC` = 2.

Graph model

From the restrictions and from the arc constraint, we deduce that we have a bijection from the successor variables to the values of interval $[1, |NODES|]$. With no explicit restrictions it would have been impossible to derive this property.

In order to express the binary constraint that links two vertices one has to make explicit the identifier of the vertices. This is why the `cycle` constraint considers objects that have two attributes:

- One fixed attribute `index`, which is the identifier of the vertex,
- One variable attribute `succ`, which is the successor of the vertex.

The graph property `NTREE = 0` is used in order to avoid having vertices which both do not belong to a circuit and have at least one successor located on a circuit. This concretely means that all vertices of the final graph should belong to a circuit.

Usage

The PhD thesis of Eric Bourreau [93] mentions the following applications of the `cycle` constraint:

- The *balanced Euler knight* problem where one tries to cover a rectangular chessboard of size $N \cdot M$ by C knights which all have to visit between $2 \cdot \lfloor \lfloor (N \cdot M) / C \rfloor / 2 \rfloor$ and $2 \cdot \lceil \lceil (N \cdot M) / C \rceil / 2 \rceil$ distinct locations. For some values of N , M and C there does not exist any solution to the previous problem. This is for instance the case when $N = M = C = 6$.
- Some *pick-up delivery* problems where a fleet of vehicles has to transport a set of orders. Each order is characterized by its initial location, its final destination and its weight. In addition one has also to take into account the capacity of the different vehicles.

Remark

In the original `cycle` constraint of CHIP the `index` attribute was not explicitly present. It was implicitly defined as the position of a variable in a list.

In an early version of the CHIP there was a constraint named `circuit` which, from a declarative point of view, was equivalent to `cycle(1, NODES)`. In ALICE [2] the `circuit` constraint was also present.

Algorithm

Since all `succ` variables have to take distinct values one can reuse the algorithms associated to the `alldifferent` constraint. A second necessary condition is to have no more than $\max(\text{NCYCLE})$ strongly connected components. Since all the vertices of a circuit belong to the same strongly connected component an arc going from one strongly connected component to another strongly connected component has to be removed.

See also

`circuit`, `cycle_card_on_path`, `cycle_resource`, `derangement`, `inverse`, `map`, `symmetric_alldifferent`, `tree`.

Key words

graph constraint, `circuit`, `cycle`, permutation, graph partitioning constraint, connected component, strongly connected component, Euler knight, pick-up delivery, `one_succ`.

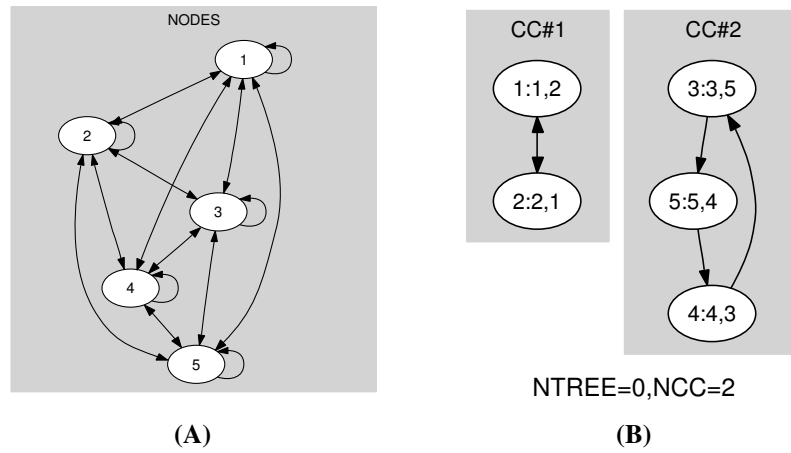


Figure 4.129: Initial and final graph of the cycle constraint

4.61 cycle_card_on_path

Origin	CHIP
Constraint	cycle_card_on_path(NCYCLE, NODES, ATLEAST, ATMOST, PATH_LEN, VALUES)
Argument(s)	NCYCLE : dvar NODES : collection(index – int, succ – dvar, colour – dvar) ATLEAST : int ATMOST : int PATH_LEN : int VALUES : collection(val – int)
Restriction(s)	$NCYCLE \geq 1$ $NCYCLE \leq NODES $ required(NODES, [index, succ, colour]) $NODES.index \geq 1$ $NODES.index \leq NODES $ distinct(NODES, index) $NODES.succ \geq 1$ $NODES.succ \leq NODES $ $ATLEAST \geq 0$ $ATLEAST \leq PATH_LEN$ $ATMOST \geq ATLEAST$ $PATH_LEN \geq 0$ required(VALUES, val) distinct(VALUES, val)
Purpose	Consider a digraph G described by the NODES collection. NCYCLE is the number of circuits for covering G in such a way that each vertex belongs to one single circuit. In addition the following constraint must also hold: On each set of PATH_LENGTH consecutive distinct vertices of each final circuit, the number of vertices for which the attribute colour takes his value in the collection of values VALUES should be located within the range [ATLEAST, ATMOST].
Arc input(s)	NODES
Arc generator	$CLIQUE \mapsto \text{collection}(\text{nodes1}, \text{nodes2})$
Arc arity	2
Arc constraint(s)	nodes1.succ = nodes2.index
Graph property(ies)	<ul style="list-style-type: none"> • NTREE = 0 • NCC = NCYCLE
Sets	$PATH_LENGTH(PATH_LEN) \mapsto$ $\left[\text{variables} - \text{col} \left(\begin{array}{l} \text{VARIABLES} - \text{collection}(\text{var} - \text{dvar}), \\ [\text{item}(\text{var} - \text{NODES.colour})] \end{array} \right) \right]$

Constraint(s) on sets among_low_up(ATLEAST, ATMOST, variables, VALUES)

Example cycle_card_on_path $2, \left(\begin{array}{l} \left\{ \begin{array}{l} \text{index} - 1 \quad \text{succ} - 7 \quad \text{colour} - 2, \\ \text{index} - 2 \quad \text{succ} - 4 \quad \text{colour} - 3, \\ \text{index} - 3 \quad \text{succ} - 8 \quad \text{colour} - 2, \\ \text{index} - 4 \quad \text{succ} - 9 \quad \text{colour} - 1, \\ \text{index} - 5 \quad \text{succ} - 1 \quad \text{colour} - 2, \\ \text{index} - 6 \quad \text{succ} - 2 \quad \text{colour} - 1, \\ \text{index} - 7 \quad \text{succ} - 5 \quad \text{colour} - 1, \\ \text{index} - 8 \quad \text{succ} - 6 \quad \text{colour} - 1, \\ \text{index} - 9 \quad \text{succ} - 3 \quad \text{colour} - 1 \end{array} \right\}, 1, 2, 3, \\ \{ \text{val} - 1 \} \end{array} \right)$

Parts (A) and (B) of Figure 4.130 respectively show the initial and final graph. Since we use the NCC graph property, we show the two connected components of the final graph. The constraint cycle_card_on_path holds since all the vertices belong to a circuit (i.e. NTREE = 0) and since for each set of three consecutive vertices, colour 1 occurs at least once and at most twice (i.e. the among_low_up constraint holds).

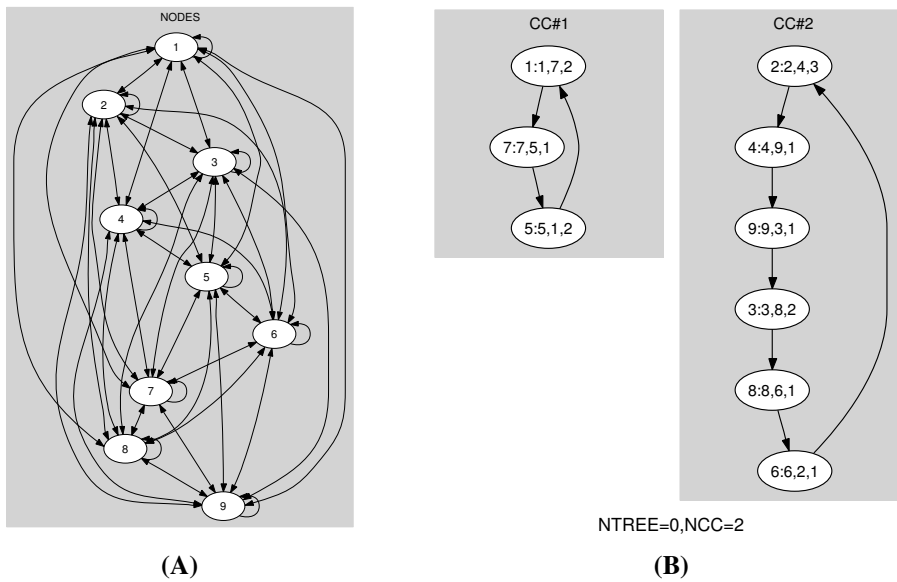


Figure 4.130: Initial and final graph of the cycle_card_on_path constraint

Usage

Assume that the vertices of G are partitioned into the following two categories:

- Clients to visit.
- Depots where one can reload a vehicle.

Using the cycle_card_on_path constraint we can express a constraint like: After visiting three consecutive clients we should visit a depot. This is typically not possible with the atmost constraint since we don't know in advance the set of variables on which to post the atmost constraint.

- Remark** This constraint is a special case of the `sequence` parameter of the `cycle` constraint of CHIP [93, pages 121–128].
- See also** `cycle`, `among_low_up`.
- Key words** graph constraint, sliding sequence constraint, sequence, connected component, coloured, `one_succ`.

20000128

393

4.62 cycle_or_accessibility

Origin	Inspired by [94].
Constraint	cycle_or_accessibility(MAXDIST, NCYCLE, NODES)
Argument(s)	MAXDIST : int NCYCLE : dvar NODES : collection(index – int, succ – dvar, x – int, y – int)
Restriction(s)	MAXDIST \geq 0 NCYCLE \geq 1 NCYCLE \leq NODES required(NODES, [index, succ, x, y]) NODES.index \geq 1 NODES.index \leq NODES distinct(NODES, index) NODES.succ \geq 0 NODES.succ \leq NODES NODES.x \geq 0 NODES.y \geq 0
Purpose	Consider a digraph G described by the NODES collection. Cover a subset of the vertices of G by a set of vertex-disjoint circuits in such a way that the following property holds: For each uncovered vertex v_1 of G there exists at least one covered vertex v_2 of G such that the Manhattan distance between v_1 and v_2 is less than or equal to MAXDIST.
Arc input(s)	NODES
Arc generator	CLIQUE \mapsto collection(nodes1, nodes2)
Arc arity	2
Arc constraint(s)	nodes1.succ = nodes2.index
Graph property(ies)	<ul style="list-style-type: none"> • NTREE = 0 • NCC = NCYCLE
Arc input(s)	NODES
Arc generator	CLIQUE \mapsto collection(nodes1, nodes2)
Arc arity	2
Arc constraint(s)	$\bigvee \left(\begin{array}{l} \text{nodes1.succ} = \text{nodes2.index}, \\ \bigwedge \left(\begin{array}{l} \text{nodes1.succ} = 0, \\ \text{nodes2.succ} \neq 0, \\ \text{abs}(\text{nodes1.x} - \text{nodes2.x}) + \text{abs}(\text{nodes1.y} - \text{nodes2.y}) \leq \text{MAXDIST} \end{array} \right) \end{array} \right)$

Graph property(ies) NVERTEX = |NODES|

Sets PRED \mapsto $\left[\begin{array}{l} \text{variables} - \text{col}(\text{VARIABLES} - \text{collection}(\text{var} - \text{dvar}), [\text{item}(\text{var} - \text{NODES.succ})]), \\ \text{destination} \end{array} \right]$

Constraint(s) on sets nvalues_except_0(variables, =, 1)

Example cycle_or_accessibility $\left(3, 2, \left\{ \begin{array}{l} \text{index} - 1 \quad \text{succ} - 6 \quad \text{x} - 4 \quad \text{y} - 5, \\ \text{index} - 2 \quad \text{succ} - 0 \quad \text{x} - 9 \quad \text{y} - 1, \\ \text{index} - 3 \quad \text{succ} - 0 \quad \text{x} - 2 \quad \text{y} - 4, \\ \text{index} - 4 \quad \text{succ} - 1 \quad \text{x} - 2 \quad \text{y} - 6, \\ \text{index} - 5 \quad \text{succ} - 5 \quad \text{x} - 7 \quad \text{y} - 2, \\ \text{index} - 6 \quad \text{succ} - 4 \quad \text{x} - 4 \quad \text{y} - 7, \\ \text{index} - 7 \quad \text{succ} - 0 \quad \text{x} - 6 \quad \text{y} - 4 \end{array} \right\} \right)$

Parts (A) and (B) of Figure 4.131 respectively show the initial and final graph associated to the second graph constraint. Figure 4.132 represents the solution associated to the previous example. The covered vertices are colored in gray while the links starting from the uncovered vertices are dashed. In the solution we have 2 circuits and 3 uncovered nodes. All the uncovered nodes are located at a distance that does not exceed 3 from at least one covered node.

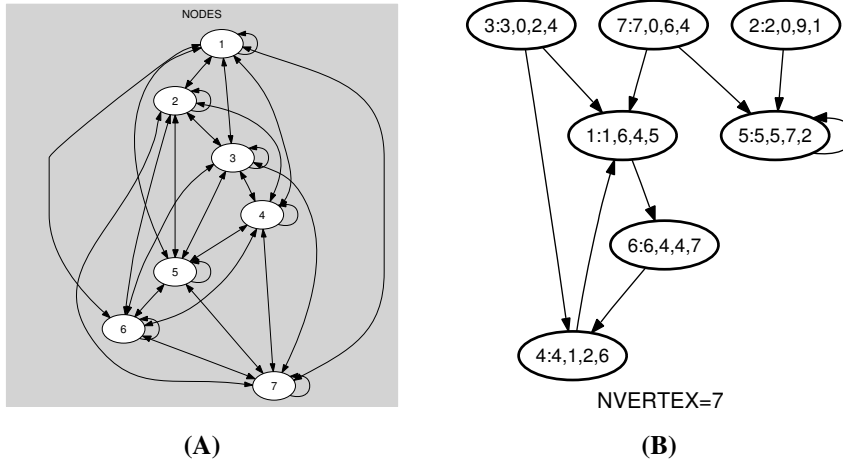


Figure 4.131: Initial and final graph of the cycle_or_accessibility constraint

Graph model

For each vertex v we have introduced the following attributes:

- **index**: The label associated to v ,
- **succ**: If v is not covered by a circuit then 0; If v is covered by a circuit then index of the successor of v .
- **x**: The x-coordinate of v ,
- **y**: The y-coordinate of v .

The first graph constraint enforces all vertices which have a non-zero successor to form a set of NCYCLE vertex-disjoint circuits.

The final graph associated to the second graph constraint contains two types of arcs:

- The arcs belonging to one circuit (i.e. $\text{nodes1.succ} = \text{nodes2.index}$),
- The arcs between one vertex v_1 that does not belong to any circuit (i.e. $\text{nodes1.succ} = 0$) and one vertex v_2 located on a circuit (i.e. $\text{nodes2.succ} \neq 0$) such that the Manhattan distance between v_1 and v_2 is less than or equal to MAXDIST.

In order to specify the fact that each vertex is involved in at least one arc we use the graph property $\text{NVERTEX} = |\text{NODES}|$. Finally the dynamic constraint $\text{nvalues_except_0}(\text{variables}, =, 1)$ expresses the fact that for each vertex v , there is exactly one predecessor of v which belong to a circuit.

Signature

Since $|\text{NODES}|$ is the maximum number of vertices of the final graph associated to the second graph constraint we can rewrite $\text{NVERTEX} = |\text{NODES}|$ to $\text{NVERTEX} \geq |\text{NODES}|$. This leads to simplify NVERTEX to NVERTEX.

Remark

This kind of facilities location problem is described in [94, pages 187–189] pages. In addition to our example they also mention the cost problem that is usually a trade-off between the vertices that are directly covered by circuits and the others.

See also

nvalues_except_0 .

Key words

graph constraint, geometrical constraint, strongly connected component, facilities location problem.

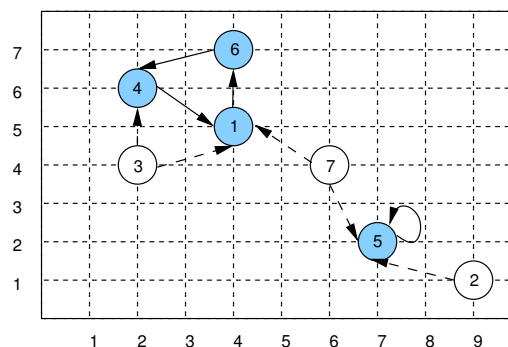


Figure 4.132: Final graph associated to the facilities location problem

20000128

397

4.63 cycle_resource

Origin	CHIP
Constraint	cycle_resource(RESOURCE, TASK)
Argument(s)	RESOURCE : collection(id - int, first_task - dvar, nb_task - dvar) TASK : collection(id - int, next_task - dvar, resource - dvar)
Restriction(s)	required(RESOURCE, [id, first_task, nb_task]) RESOURCE.id ≥ 1 RESOURCE.id \leq RESOURCE distinct(RESOURCE, id) RESOURCE.first_task ≥ 1 RESOURCE.first_task \leq RESOURCE + TASK RESOURCE.nb_task ≥ 0 RESOURCE.nb_task \leq TASK required(TASK, [id, next_task, resource]) TASK.id $>$ RESOURCE TASK.id \leq RESOURCE + TASK distinct(TASK, id) TASK.next_task ≥ 1 TASK.next_task \leq RESOURCE + TASK TASK.resource ≥ 1 TASK.resource \leq RESOURCE

Purpose

Consider a digraph G defined as follows:

- To each item of the RESOURCE and TASK collections corresponds one vertex of G . A vertex that was generated from an item of the RESOURCE (respectively TASK) collection is called a *resource* vertex (respectively *task* vertex).
- There is an arc from a resource vertex r to a task vertex t if $t \in \text{RESOURCE}[r].\text{first_task}$.
- There is an arc from a task vertex t to a resource vertex r if $r \in \text{TASK}[t].\text{next_task}$.
- There is an arc from a task vertex t_1 to a task vertex t_2 if $t_2 \in \text{TASK}[t_1].\text{next_task}$.
- There is no arc between two resource vertices.

Enforce to cover G in such a way that each vertex belongs to one single circuit. Each circuit is made up from one single *resource* vertex and zero, one, or more *task* vertices. For each resource-vertex a domain variable indicates how many task-vertices belong to the corresponding circuit. For each task a domain variable gives the identifier of the resource which can effectively handle that task.

Derived Collection(s)	col $\left(\begin{array}{l} \text{RESOURCE_TASK} - \text{collection}(\text{index} - \text{int}, \text{succ} - \text{dvar}, \text{name} - \text{dvar}), \\ \left[\begin{array}{l} \text{item}(\text{index} - \text{RESOURCE.id}, \text{succ} - \text{RESOURCE.first_task}, \text{name} - \text{RESOURCE.id}), \\ \text{item}(\text{index} - \text{TASK.id}, \text{succ} - \text{TASK.next_task}, \text{name} - \text{TASK.resource}) \end{array} \right] \end{array} \right)$
Arc input(s)	RESOURCE_TASK

Arc generator	$CLIQUE \mapsto \text{collection}(\text{resource_task1}, \text{resource_task2})$
Arc arity	2
Arc constraint(s)	<ul style="list-style-type: none"> • $\text{resource_task1.succ} = \text{resource_task2.index}$ • $\text{resource_task1.name} = \text{resource_task2.name}$
Graph property(ies)	<ul style="list-style-type: none"> • $N_{TREE} = 0$ • $N_{CC} = \text{RESOURCE}$ • $N_{VERTEX} = \text{RESOURCE} + \text{TASK}$ <hr/> <p>For all items of RESOURCE:</p> <hr/>
Arc input(s)	RESOURCE_TASK
Arc generator	$CLIQUE \mapsto \text{collection}(\text{resource_task1}, \text{resource_task2})$
Arc arity	2
Arc constraint(s)	<ul style="list-style-type: none"> • $\text{resource_task1.succ} = \text{resource_task2.index}$ • $\text{resource_task1.name} = \text{resource_task2.name}$ • $\text{resource_task1.name} = \text{RESOURCE.id}$
Graph property(ies)	$N_{VERTEX} = \text{RESOURCE.nb_task} + 1$

Example	$\text{cycle_resource} \left(\left(\left\{ \begin{array}{l} \text{id} - 1 \quad \text{first_task} - 5 \quad \text{nb_task} - 3, \\ \text{id} - 2 \quad \text{first_task} - 2 \quad \text{nb_task} - 0, \\ \text{id} - 3 \quad \text{first_task} - 8 \quad \text{nb_task} - 2 \end{array} \right\}, \right. \right. \\ \left. \left. \left\{ \begin{array}{l} \text{id} - 4 \quad \text{next_task} - 7 \quad \text{resource} - 1, \\ \text{id} - 5 \quad \text{next_task} - 4 \quad \text{resource} - 1, \\ \text{id} - 6 \quad \text{next_task} - 3 \quad \text{resource} - 3, \\ \text{id} - 7 \quad \text{next_task} - 1 \quad \text{resource} - 1, \\ \text{id} - 8 \quad \text{next_task} - 6 \quad \text{resource} - 3 \end{array} \right\} \right) \right)$
----------------	---

Part (A) of Figure 4.133 shows the initial graphs (of the second graph constraint) associated to resources 1, 2 and 3. Part (B) of Figure 4.133 shows the final graphs (of the second graph constraint) associated to resources 1, 2 and 3. Since we use the **NVERTEX** graph property, the vertices of the final graphs are stressed in bold. To each resource corresponds a circuit of respectively 3, 0 and 2 task-vertices.

Graph model The graph model of the `cycle_resource` constraint illustrates the following points:

- How to differentiate the constraint on the length of a circuit according to a resource that is assigned to a circuit? This is achieved by introducing a collection of resources and by asking a different graph property for each item of that collection.
- How to introduce the concept of name which corresponds to the resource that handle a given task? This is done by adding to the arc constraint associated to the `cycle` constraint the condition that the name variables of two consecutive vertices should be equal.

Signature	Since the initial graph of the first graph constraint contains $ \text{RESOURCE} + \text{TASK} $ vertices, the corresponding final graph cannot have more than $ \text{RESOURCE} + \text{TASK} $ vertices. Therefore we can rewrite the graph property $\text{NVERTEX} = \text{RESOURCE} + \text{TASK} $ to $\text{NVERTEX} \geq \text{RESOURCE} + \text{TASK} $ and simplify <u>NVERTEX</u> to <u>NVERTEX</u> .
Usage	This constraint is useful for some vehicles routing problem where the number of locations to visit depends of the vehicle type that is effectively used. The resource attribute allows expressing various constraints such as: <ul style="list-style-type: none"> • The compatibility or incompatibility between tasks and vehicles, • The fact that certain tasks should be performed by the same vehicle, • The preassignment of certain tasks to a given vehicle.
Remark	This constraint could be expressed with the <code>cycle</code> constraint of CHIP by using the following optional parameters: <ul style="list-style-type: none"> • The <i>resource node</i> parameter [93, page 97], • The <i>circuit weight</i> parameter [93, page 101], • The <i>name</i> parameter [93, page 104].
See also	<code>cycle</code> .
Key words	graph constraint, resource constraint, graph partitioning constraint, connected component, strongly connected component, derived collection.

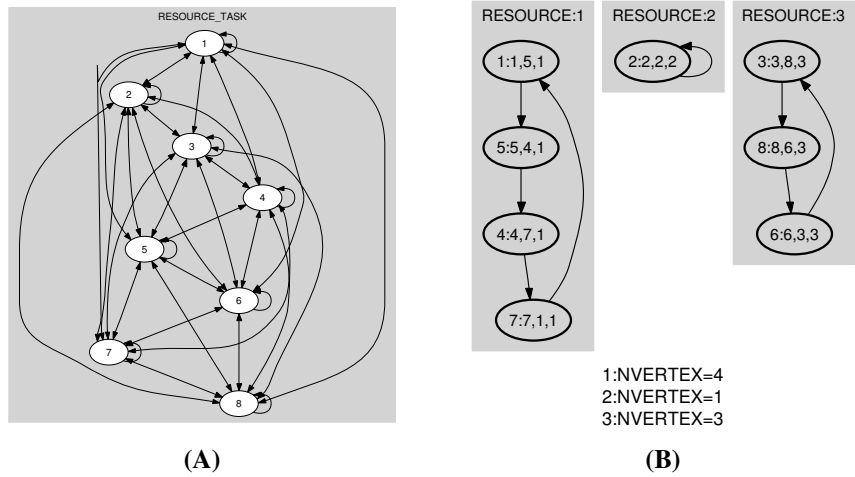


Figure 4.133: Initial and final graph of the cycle_resource constraint

4.64 cyclic_change

Origin	Derived from change.
Constraint	<code>cyclic_change(NCHANGE, CYCLE_LENGTH, VARIABLES, CTR)</code>
Argument(s)	<pre>NCHANGE : dvar CYCLE_LENGTH : int VARIABLES : collection(var - dvar) CTR : atom</pre>
Restriction(s)	<pre>NCHANGE ≥ 0 NCHANGE < VARIABLES CYCLE_LENGTH > 0 required(VARIABLES, var) CTR ∈ [=, ≠, <, ≥, >, ≤]</pre>
Purpose	<p>NCHANGE is the number of times that constraint $((X + 1) \bmod \text{CYCLE_LENGTH}) \text{CTR } Y$ holds; X and Y correspond to consecutive variables of the collection VARIABLES.</p>
Arc input(s)	VARIABLES
Arc generator	$PATH \mapsto \text{collection}(\text{variables1}, \text{variables2})$
Arc arity	2
Arc constraint(s)	$(\text{variables1.var} + 1) \bmod \text{CYCLE_LENGTH} \text{CTR variables2.var}$
Graph property(ies)	NARC = NCHANGE

Example $\text{cyclic_change} \left(2, 4, \left\{ \begin{array}{l} \text{var} - 3, \\ \text{var} - 0, \\ \text{var} - 2, \\ \text{var} - 3, \\ \text{var} - 1 \end{array} \right\}, \neq \right)$

In the previous example we have the two following changes:

- A first change between 0 and 2,
- A second change between 3 and 1.

However, the sequence 3 0 does not correspond to a change since $(3 + 1) \bmod 4$ is equal to 0. Parts (A) and (B) of Figure 4.134 respectively show the initial and final graph. Since we use the **NARC** graph property, the arcs of the final graph are stressed in bold.

Automaton Figure 4.135 depicts the automaton associated to the `cyclic_change` constraint. To each pair of consecutive variables $(\text{VAR}_i, \text{VAR}_{i+1})$ of the collection VARIABLES corresponds a 0-1 signature variable S_i . The following signature constraint links VAR_i , VAR_{i+1} and S_i : $((\text{VAR}_i + 1) \bmod \text{CYCLE_LENGTH}) \text{CTR } \text{VAR}_{i+1} \Leftrightarrow S_i$.

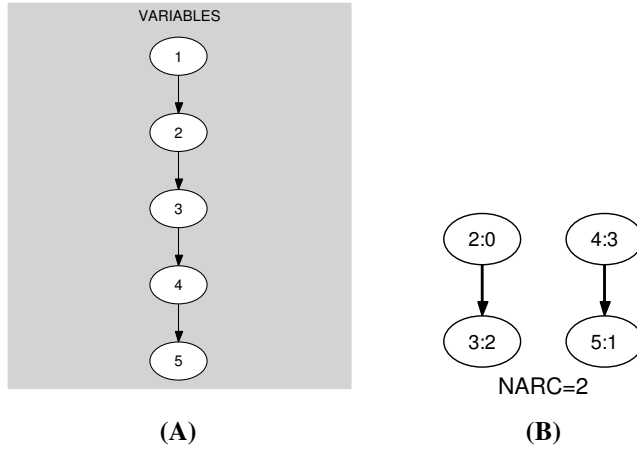


Figure 4.134: Initial and final graph of the cyclic_change constraint

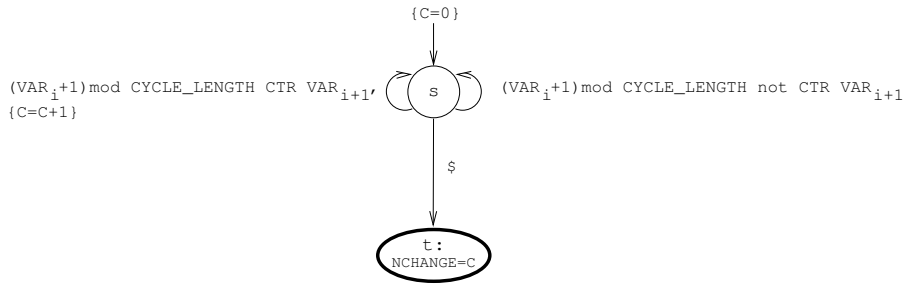


Figure 4.135: Automaton of the cyclic_change constraint

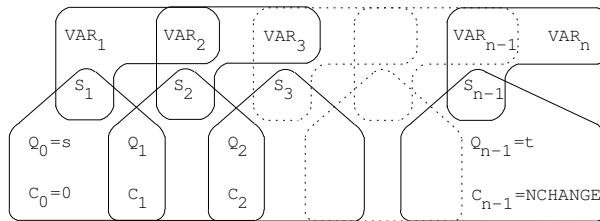


Figure 4.136: Hypergraph of the reformulation corresponding to the automaton of the cyclic_change constraint

- Usage** This constraint may be used for personnel cyclic timetabling problems where each person has to work according to cycles. In this context each variable of the `VARIABLES` collection corresponds to the type of work a person performs on a specific day. Because of some perturbation (e.g. illness, unavailability, variation of the workload) it is in practice not reasonable to ask for perfect cyclic solutions. One alternative is to use the `cyclic_change` constraint and to ask for solutions where one tries to minimize the number of cycle breaks (i.e. the variable `NCHANGE`).
- See also** `change`.
- Key words** timetabling constraint, number of changes, cyclic, automaton, automaton with counters, sliding cyclic(1) constraint network(2), acyclic, no_loop.

20000128

405

4.65 cyclic_change_joker

Origin	Derived from cyclic_change.
Constraint	cyclic_change_joker(NCHANGE, CYCLE_LENGTH, VARIABLES, CTR)
Argument(s)	NCHANGE : dvar CYCLE_LENGTH : int VARIABLES : collection(var - dvar) CTR : atom

Restriction(s)	$NCHANGE \geq 0$ $NCHANGE < VARIABLES $ required(VARIABLES, var) $CYCLE_LENGTH > 0$ $CTR \in [=, \neq, <, \geq, >, \leq]$
-----------------------	---

Purpose

NCHANGE is the number of times that the following constraint holds:

$$((X + 1) \bmod CYCLE_LENGTH) CTR Y \wedge X < CYCLE_LENGTH \wedge Y < CYCLE_LENGTH$$

X and *Y* correspond to consecutive variables of the collection VARIABLES.

Arc input(s)	VARIABLES
Arc generator	<i>PATH</i> \mapsto collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	<ul style="list-style-type: none"> • (variables1.var + 1) mod CYCLE_LENGTH CTR variables2.var • variables1.var < CYCLE_LENGTH • variables2.var < CYCLE_LENGTH
Graph property(ies)	NARC = NCHANGE

Example	$\text{cyclic_change_joker} \left(2, 4, \left\{ \begin{array}{l} \text{var} - 3, \\ \text{var} - 0, \\ \text{var} - 2, \\ \text{var} - 4, \\ \text{var} - 4, \\ \text{var} - 4, \\ \text{var} - 3, \\ \text{var} - 1, \\ \text{var} - 4 \end{array} \right\}, \neq \right)$
----------------	--

In the previous example we have the two following changes:

- A first change between 0 and 2,
- A second change between 3 and 1.

But when the joker value 4 is involved, there is no change. This is why no change is counted between values 2 and 4, between 4 and 4 and between 1 and 4. Parts (A) and (B) of Figure 4.137 respectively show the initial and final graph. Since we use the **NARC** graph property, the arcs of the final graph are stressed in bold.

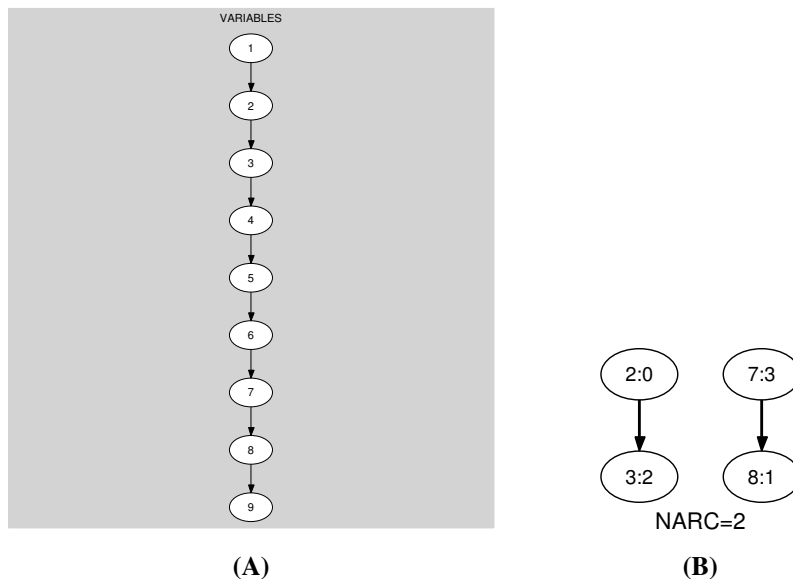


Figure 4.137: Initial and final graph of the `cyclic_change_joker` constraint

Graph model

The *joker values* are those values that are greater than or equal to `CYCLE_LENGTH`. We do not count any change for those arc constraints involving at least one variable taking a joker value.

Automaton

Figure 4.138 depicts the automaton associated to the `cyclic_change_joker` constraint. To each pair of consecutive variables (VAR_i, VAR_{i+1}) of the collection `VARIABLES` corresponds a 0-1 signature variable S_i . The following signature constraint links VAR_i , VAR_{i+1} and S_i :

$$\begin{aligned} &(((VAR_i + 1) \bmod \text{CYCLE_LENGTH}) \text{CTR } VAR_{i+1} \wedge \\ & (VAR_i < \text{CYCLE_LENGTH}) \wedge (VAR_{i+1} < \text{CYCLE_LENGTH})) \Leftrightarrow S_i. \end{aligned}$$

Usage

The `cyclic_change_joker` constraint can be used in the same context as the `cycle_change` constraint with the additional feature: In our example codes 0 to 3 correspond to different type of activities (i.e. working the morning, the afternoon or the night) and code 4 represents a holiday. We want to express the fact that we don't count any change for two consecutive days d_1, d_2 such that d_1 or d_2 is a holiday.

See also

`change`.

Key words

timetabling constraint, number of changes, cyclic, joker value, automaton, automaton with counters, sliding cyclic(1) constraint network(2), acyclic, no_loop.

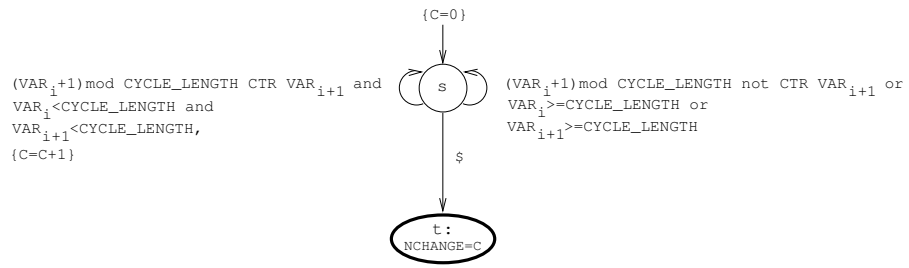


Figure 4.138: Automaton of the cyclic_change_joker constraint

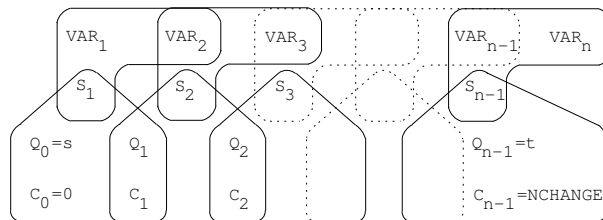


Figure 4.139: Hypergraph of the reformulation corresponding to the automaton of the cyclic_change_joker constraint

20000128

409

4.66 decreasing

Origin	Inspired by increasing.
Constraint	<code>decreasing(VARIABLES)</code>
Argument(s)	<code>VARIABLES : collection(var - dvar)</code>
Restriction(s)	<code> VARIABLES > 0</code> <code>required(VARIABLES, var)</code>
Purpose	The variables of the collection VARIABLEs are decreasing.
<hr/>	
Arc input(s)	VARIABLES
Arc generator	$\text{PATH} \mapsto \text{collection}(\text{variables1}, \text{variables2})$
Arc arity	2
Arc constraint(s)	$\text{variables1.var} \geq \text{variables2.var}$
Graph property(ies)	$\text{NARC} = \text{VARIABLES} - 1$
<hr/>	
Example	<code>decreasing({var - 8, var - 4, var - 1, var - 1})</code>

Parts (A) and (B) of Figure 4.140 respectively show the initial and final graph. Since we use the **NARC** graph property, the arcs of the final graph are stressed in bold.

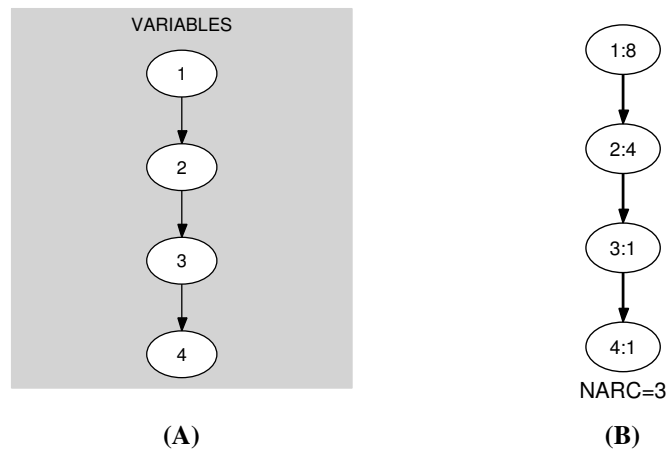


Figure 4.140: Initial and final graph of the decreasing constraint

- Automaton** Figure 4.141 depicts the automaton associated to the `decreasing` constraint. To each pair of consecutive variables (VAR_i, VAR_{i+1}) of the collection `VARIABLES` corresponds a 0-1 signature variable S_i . The following signature constraint links VAR_i , VAR_{i+1} and S_i : $VAR_i < VAR_{i+1} \Leftrightarrow S_i$.
- See also** `strictly_decreasing`, `increasing`, `strictly_increasing`.
- Key words** decomposition, order constraint, automaton, automaton without counters, sliding cyclic(1) constraint network(1).

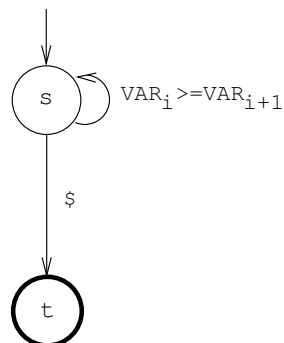


Figure 4.141: Automaton of the decreasing constraint

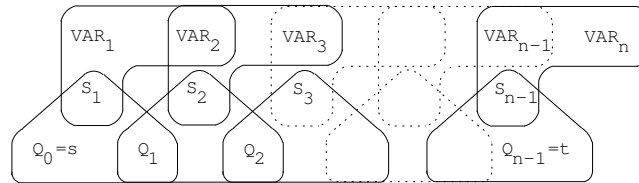


Figure 4.142: Hypergraph of the reformulation corresponding to the automaton of the decreasing constraint

20040814

413

4.67 deepest_valley

Origin	Derived from valley.
Constraint	deepest_valley(DEPTH, VARIABLES)
Argument(s)	DEPTH : dvar VARIABLES : collection(var - dvar)
Restriction(s)	DEPTH ≥ 0 VARIABLES.var ≥ 0 required(VARIABLES, var)

Purpose

A variable V_k ($1 < k < m$) of the sequence of variables $VARIABLES = V_1, \dots, V_m$ is a *valley* if and only if there exist an i ($1 < i \leq k$) such that $V_{i-1} > V_i$ and $V_i = V_{i+1} = \dots = V_k$ and $V_k < V_{k+1}$. DEPTH is the minimum value of the valley variables. If no such variable exists DEPTH is equal to the default value MAXINT.

Example

$$\text{deepest_valley} \left(2, \left\{ \begin{array}{l} \text{var} - 5, \\ \text{var} - 3, \\ \text{var} - 4, \\ \text{var} - 8, \\ \text{var} - 8, \\ \text{var} - 2, \\ \text{var} - 7, \\ \text{var} - 1 \end{array} \right\} \right)$$

The previous constraint holds since 2 is the deepest valley of the sequence 5 3 4 8 8 2 7 1.

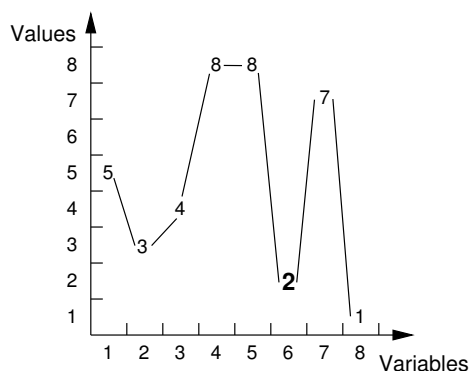


Figure 4.143: The sequence and its deepest valley

Automaton

Figure 4.144 depicts the automaton associated to the deepest_valley constraint. To each pair of consecutive variables (VAR_i, VAR_{i+1}) of the collection VARIABLES corresponds a signature variable S_i . The following signature constraint links VAR_i, VAR_{i+1} and S_i :

$$VAR_i < VAR_{i+1} \Leftrightarrow S_i = 0 \wedge VAR_i = VAR_{i+1} \Leftrightarrow S_i = 1 \wedge VAR_i > VAR_{i+1} \Leftrightarrow S_i = 2.$$

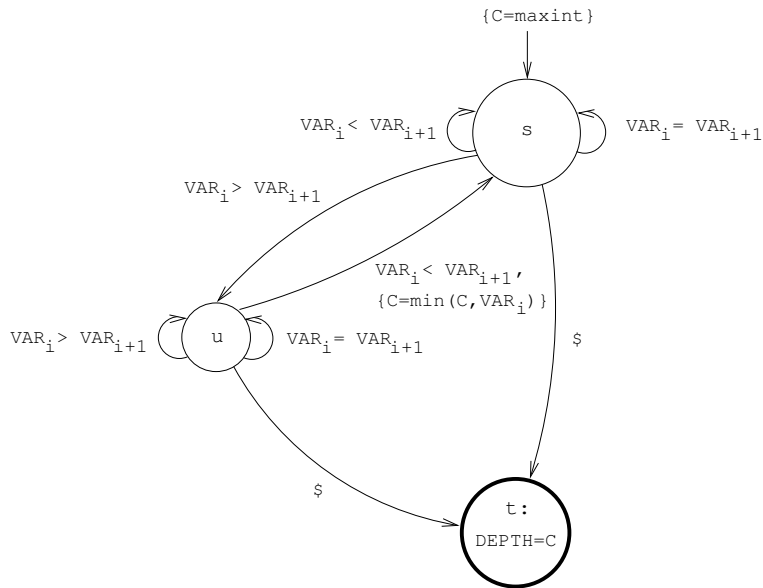


Figure 4.144: Automaton of the deepest_valley constraint

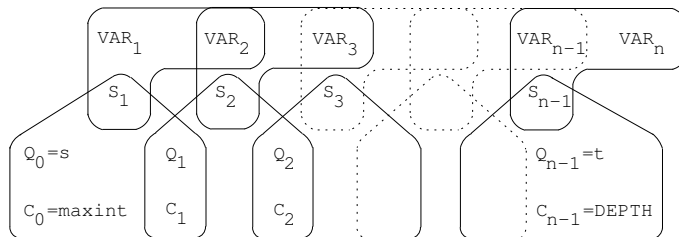


Figure 4.145: Hypergraph of the reformulation corresponding to the automaton of the deepest_valley constraint

See also valley, heighest_peak.

Key words sequence, maxint, automaton, automaton with counters,
sliding cyclic(1) constraint network(2).

20040530

417

4.68 derangement

Origin	Derived from <code>cycle</code> .
Constraint	<code>derangement(NODES)</code>
Argument(s)	<code>NODES : collection(index - int, succ - dvar)</code>
Restriction(s)	<pre> required(NODES, [index, succ]) NODES.index ≥ 1 NODES.index ≤ NODES distinct(NODES, index) NODES.succ ≥ 1 NODES.succ ≤ NODES </pre>
Purpose	Enforce to have a permutation with no cycle of length one. The permutation is depicted by the <code>succ</code> attribute of the <code>NODES</code> collection.
<hr/>	
Arc input(s)	<code>NODES</code>
Arc generator	<code>CLIQUE</code> \mapsto <code>collection(nodes1, nodes2)</code>
Arc arity	2
Arc constraint(s)	<ul style="list-style-type: none"> • <code>nodes1.succ = nodes2.index</code> • <code>nodes1.succ ≠ nodes1.index</code>
Graph property(ies)	<code>N</code> TREE = 0
<hr/>	
Example	$\text{derangement} \left(\left(\begin{array}{l} \left(\begin{array}{ll} \text{index} - 1 & \text{succ} - 2, \\ \text{index} - 2 & \text{succ} - 1, \\ \text{index} - 3 & \text{succ} - 5, \\ \text{index} - 4 & \text{succ} - 3, \\ \text{index} - 5 & \text{succ} - 4 \end{array} \right) \right) \right)$ <p>In the permutation of the previous example we have the following 2 cycles: $1 \rightarrow 2 \rightarrow 1$ and $3 \rightarrow 5 \rightarrow 4 \rightarrow 3$. Parts (A) and (B) of Figure 4.146 respectively show the initial and final graph. The constraint holds since the final graph does not contain any vertex which do not belong to a circuit (i.e. <code>N</code>TREE = 0).</p>
Graph model	<p>In order to express the binary constraint that links two vertices of the <code>NODES</code> collection one has to make explicit the index value of the vertices. This is why the <code>derangement</code> constraint considers objects that have two attributes:</p> <ul style="list-style-type: none"> • One fixed attribute <code>index</code>, which is the identifier of the vertex, • One variable attribute <code>succ</code>, which is the successor of the vertex. <p>Forbidding cycles of length one is achieved by the second condition of the arc constraint.</p>

- Signature** Since 0 is the smallest possible value of **NTREE** we can rewrite the graph property $\mathbf{NTREE} = 0$ to $\mathbf{NTREE} \leq 0$. This leads to simplify **NTREE** to **NTREE**.
- Remark** A special case of the `cycle [37]` constraint.
- See also** `alldifferent`, `cycle`.
- Key words** graph constraint, permutation.

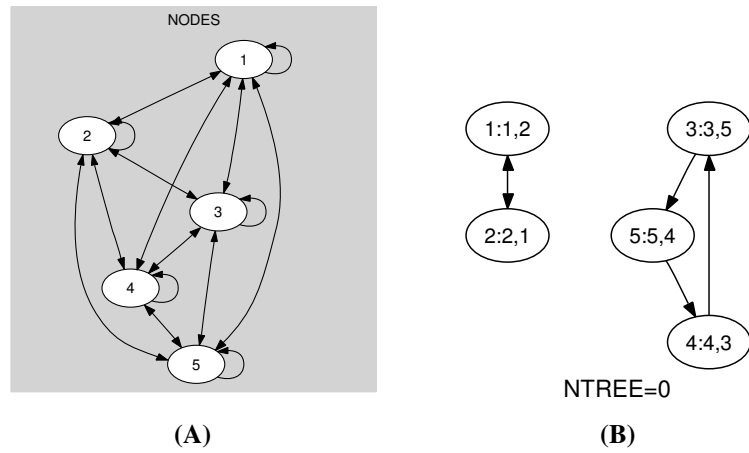


Figure 4.146: Initial and final graph of the derangement constraint

20000128

421

4.69 differ_from_at_least_k_pos

Origin	Inspired by [56].
Constraint	<code>differ_from_at_least_k_pos(K, VECTOR1, VECTOR2)</code>
Type(s)	<code>VECTOR : collection(var - dvar)</code>
Argument(s)	<code>K : int</code> <code>VECTOR1 : VECTOR</code> <code>VECTOR2 : VECTOR</code>
Restriction(s)	<code>required(VECTOR, var)</code> $K \geq 0$ $K \leq \text{VECTOR1} $ $ \text{VECTOR1} = \text{VECTOR2} $
Purpose	Enforce two vectors VECTOR1 and VECTOR2 to differ from at least K positions.
<hr/>	
Arc input(s)	VECTOR1 VECTOR2
Arc generator	$\text{PRODUCT}(=) \mapsto \text{collection}(\text{vector1}, \text{vector2})$
Arc arity	2
Arc constraint(s)	<code>vector1.var \neq vector2.var</code>
Graph property(ies)	$\text{NARC} \geq K$

Example	$\text{differ_from_at_least_k_pos} \left(2, \left\{ \begin{array}{l} \text{var} - 2, \\ \text{var} - 5, \\ \text{var} - 2, \\ \text{var} - 0 \end{array} \right\}, \left\{ \begin{array}{l} \text{var} - 3, \\ \text{var} - 6, \\ \text{var} - 2, \\ \text{var} - 1 \end{array} \right\} \right)$
----------------	---

The previous constraint holds since the first and second vectors differ from 3 positions which is greater than or equal to $K = 2$. Parts (A) and (B) of Figure 4.147 respectively show the initial and final graph. Since we use the **NARC** graph property, the arcs of the final graph are stressed in bold.

Automaton	Figure 4.148 depicts the automaton associated to the <code>differ_from_at_least_k_pos</code> constraint. Let VAR1_i and VAR2_i be the i^{th} variables of the VECTOR1 and VECTOR2 collections. To each pair of variables $(\text{VAR1}_i, \text{VAR2}_i)$ corresponds a signature variable S_i . The following signature constraint links $\text{VAR1}_i, \text{VAR2}_i$ and S_i : $\text{VAR1}_i = \text{VAR2}_i \Leftrightarrow S_i$.
------------------	---

Remark	Used in the Arc constraint(s) slot of the <code>all_differ_from_at_least_k_pos</code> constraint.
---------------	--

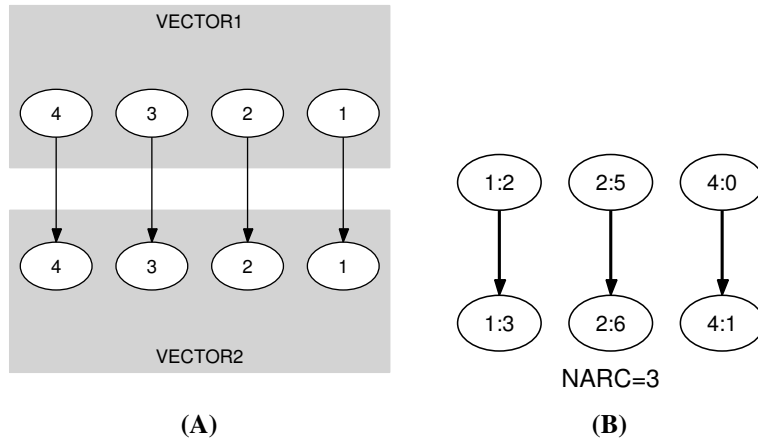


Figure 4.147: Initial and final graph of the differ_from_at_least_k_pos constraint

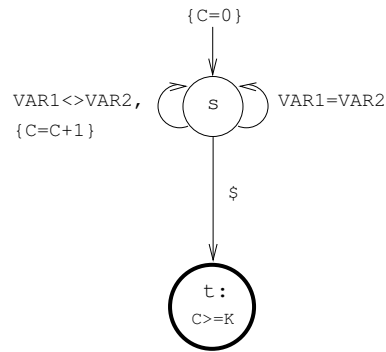


Figure 4.148: Automaton of the differ_from_at_least_k_pos constraint

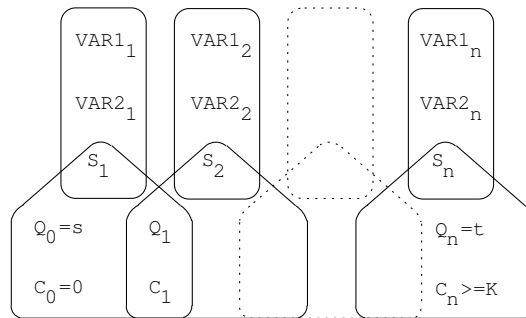


Figure 4.149: Hypergraph of the reformulation corresponding to the automaton of the differ_from_at_least_k_pos constraint

Used in *all_differ_from_at_least_k_pos.*

Key words value constraint, vector, automaton, automaton with counters,
alpha-acyclic constraint network(2).

20030820

425

4.70 diffn

Origin	[37]
Constraint	diffn(ORTHOTOPES)
Type(s)	ORTHOTOPE : collection(ori - dvar, siz - dvar, end - dvar)
Argument(s)	ORTHOTOPES : collection(orth - ORTHOTOPE)
Restriction(s)	$ \text{ORTHOTOPE} > 0$ $\text{require_at_least}(2, \text{ORTHOTOPE}, [\text{ori}, \text{siz}, \text{end}])$ $\text{ORTHOTOPE.siz} \geq 0$ $\text{required}(\text{ORTHOTOPES}, \text{orth})$ $\text{same_size}(\text{ORTHOTOPES}, \text{orth})$
Purpose	Generalized multi-dimensional non-overlapping constraint: Holds if, for each pair of orthotopes (O_1, O_2) , O_1 and O_2 do not overlap. Two orthotopes do not overlap if there exists at least one dimension where their projections do not overlap.
Arc input(s)	ORTHOTOPES
Arc generator	$\text{SELF} \mapsto \text{collection}(\text{orthotopes})$
Arc arity	1
Arc constraint(s)	orth_link_ori_siz_end(orthotopes.orth)
Graph property(ies)	$\text{NARC} = \text{ORTHOTOPES} $
Arc input(s)	ORTHOTOPES
Arc generator	$\text{CLIQUE}(\neq) \mapsto \text{collection}(\text{orthotopes1}, \text{orthotopes2})$
Arc arity	2
Arc constraint(s)	two_orth_do_not_overlap(orthotopes1.orth, orthotopes2.orth)
Graph property(ies)	$\text{NARC} = \text{ORTHOTOPES} * \text{ORTHOTOPES} - \text{ORTHOTOPES} $
Example	$\text{diffn} \left(\left(\left\{ \begin{array}{l} \text{orth} - \left\{ \begin{array}{l} \text{ori} - 2 \quad \text{siz} - 2 \quad \text{end} - 4, \\ \text{ori} - 1 \quad \text{siz} - 3 \quad \text{end} - 4 \end{array} \right\}, \\ \text{orth} - \left\{ \begin{array}{l} \text{ori} - 4 \quad \text{siz} - 4 \quad \text{end} - 8, \\ \text{ori} - 3 \quad \text{siz} - 3 \quad \text{end} - 3 \end{array} \right\}, \\ \text{orth} - \left\{ \begin{array}{l} \text{ori} - 9 \quad \text{siz} - 2 \quad \text{end} - 11, \\ \text{ori} - 4 \quad \text{siz} - 3 \quad \text{end} - 7 \end{array} \right\} \end{array} \right\} \right) \right)$

Parts (A) and (B) of Figure 4.150 respectively show the initial and final graph associated to the second graph constraint. Since we use the **NARC** graph property, the arcs of the final graph are stressed in bold. Figure 4.151 represents the respective position of the three rectangles of the example. The coordinates of the leftmost lowest corner of each rectangle are stressed in bold.

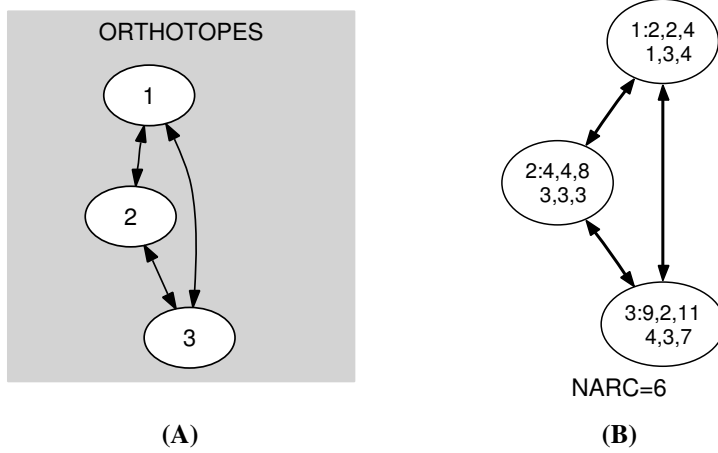


Figure 4.150: Initial and final graph of the `diffn` constraint

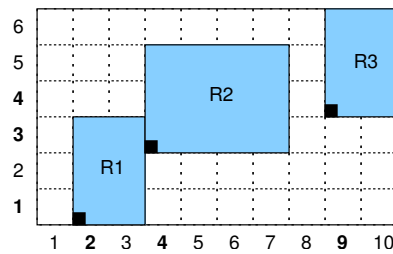



Figure 4.151: The three rectangles of the example

Graph model	The <code>diffn</code> constraint is expressed by using two graph constraints: <ul style="list-style-type: none"> • The first graph constraint enforces for each dimension and for each orthotope the link between the corresponding <code>ori</code>, <code>siz</code> and <code>end</code> attributes. • The second graph constraint imposes each pair of distinct orthotopes to not overlap.
Signature	Since $ \text{ORTHOTOPES} $ is the maximum number of vertices of the final graph of the first graph constraint we can rewrite $\text{NARC} = \text{ORTHOTOPES} $ to $\text{NARC} \geq \text{ORTHOTOPES} $. This leads to simplify $\overline{\text{NARC}}$ to $\overline{\text{NARC}}$. Since we use the $\text{CLIQUE}(\neq)$ arc generator on the <code>ORTHOTOPES</code> collection, $ \text{ORTHOTOPES} \cdot \text{ORTHOTOPES} - \text{ORTHOTOPES} $ is the maximum number of vertices of the final graph of the second graph constraint. Therefore we can rewrite $\text{NARC} = \text{ORTHOTOPES} \cdot \text{ORTHOTOPES} - \text{ORTHOTOPES} $ to $\text{NARC} \geq \text{ORTHOTOPES} \cdot \text{ORTHOTOPES} - \text{ORTHOTOPES} $. Again, this leads to simplify $\overline{\text{NARC}}$ to $\overline{\text{NARC}}$.
Usage	The <code>diffn</code> constraint occurs in placement and scheduling problems. It was for instance used for scheduling problems where one has to both assign each non-preemptive task to a resource and fix its origin so that two tasks which are assigned to the same resource do not overlap. A practical application from the area of the design of memory-dominated embedded systems [95] can be found in [96].
Algorithm	For the two-dimensional case of <code>diffn</code> a possible filtering algorithm based on <i>sweep</i> is described in [97]. For the n -dimensional case of <code>diffn</code> a filtering algorithm handling the fact that two objects do not overlap is given in [98]. Extensions of the non-overlapping constraint to polygons and to more complex shapes are respectively described in [98] and in [99]. Specialized propagation algorithms for the <i>squared squares</i> problem [100] (based on the fact that no waste is permitted) are given in [101] and in [102].
Used in	<code>diffn_column</code> , <code>diffn_include</code> , <code>place_in_pyramid</code> .
See also	<code>orth_link_ori_siz_end</code> , <code>two_orth_do_not_overlap</code> .
Key words	decomposition, geometrical constraint, orthotope, polygon, non-overlapping, sweep, squared squares.

20000128

429

4.71 `diffn_column`

Origin	CHIP: option guillotine cut (column) of <code>diffn</code> .
Constraint	<code>diffn_column(ORTHOTOPES, N)</code>
Type(s)	<code>ORTHOTOPE</code> : <code>collection(ori - dvar, siz - dvar, end - dvar)</code>
Argument(s)	<code>ORTHOTOPES</code> : <code>collection(orth - ORTHOTOPE)</code> <code>N</code> : <code>int</code>
Restriction(s)	<code> ORTHOTOPE > 0</code> <code>require_at_least(2, ORTHOTOPE, [ori, siz, end])</code> <code>ORTHOTOPE.siz ≥ 0</code> <code>required(ORTHOTOPES, orth)</code> <code>same_size(ORTHOTOPES, orth)</code> <code>N > 0</code> <code>N ≤ ORTHOTOPE </code> <code>diffn(ORTHOTOPES)</code>
Purpose	
Arc input(s)	<code>ORTHOTOPES</code>
Arc generator	$\text{CLIQUE}(<) \mapsto \text{collection}(\text{orthotopes1}, \text{orthotopes2})$
Arc arity	2
Arc constraint(s)	<code>two_orth_column(orthotopes1.orth, orthotopes2.orth, N)</code>
Graph property(ies)	$\overline{\text{NARC}} = \text{ORTHOTOPES} * (\text{ORTHOTOPES} - 1) / 2$
Example	$\text{diffn_column} \left(\left(\left\{ \begin{array}{l} \text{orth} - \left\{ \begin{array}{lll} \text{ori} - 1 & \text{siz} - 3 & \text{end} - 4, \\ \text{ori} - 1 & \text{siz} - 1 & \text{end} - 2 \end{array} \right\}, \\ \text{orth} - \left\{ \begin{array}{lll} \text{ori} - 4 & \text{siz} - 2 & \text{end} - 6, \\ \text{ori} - 1 & \text{siz} - 3 & \text{end} - 4 \end{array} \right\} \end{array} \right\}, 1 \right)$
See also	<code>diffn</code> , <code>two_orth_column</code> , <code>diffn_include</code> .
Key words	decomposition, geometrical constraint, positioning constraint, orthotope, guillotine cut.

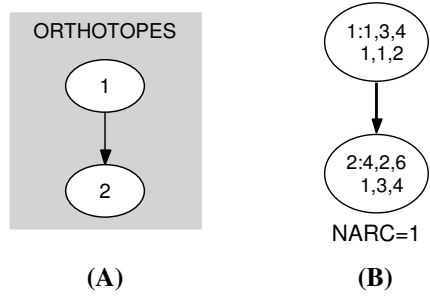



Figure 4.152: Initial and final graph of the `diffn_column` constraint

4.72 `diffn_include`

Origin	CHIP: option guillotine cut (include) of <code>diffn</code> .
Constraint	<code>diffn_include(ORTHOTOPES, N)</code>
Type(s)	<code>ORTHOTOPE</code> : <code>collection(ori - dvar, siz - dvar, end - dvar)</code>
Argument(s)	<code>ORTHOTOPES</code> : <code>collection(orth - ORTHOTOPE)</code> <code>N</code> : <code>int</code>
Restriction(s)	<code> ORTHOTOPE > 0</code> <code>require_at_least(2, ORTHOTOPE, [ori, siz, end])</code> <code>ORTHOTOPE.siz ≥ 0</code> <code>required(ORTHOTOPES, orth)</code> <code>same_size(ORTHOTOPES, orth)</code> <code>N > 0</code> <code>N ≤ ORTHOTOPE </code> <code>diffn(ORTHOTOPES)</code>
Purpose	
Arc input(s)	<code>ORTHOTOPES</code>
Arc generator	$\text{CLIQUE}(<) \mapsto \text{collection}(\text{orthotopes1}, \text{orthotopes2})$
Arc arity	2
Arc constraint(s)	<code>two_orth_include(orthotopes1.orth, orthotopes2.orth, N)</code>
Graph property(ies)	$\overline{\text{NARC}} = \text{ORTHOTOPES} * (\text{ORTHOTOPES} - 1) / 2$
Example	$\text{diffn_include} \left(\left(\left\{ \begin{array}{l} \text{orth} - \left\{ \begin{array}{l} \text{ori} - 1 \quad \text{siz} - 3 \quad \text{end} - 4, \\ \text{ori} - 1 \quad \text{siz} - 1 \quad \text{end} - 2 \end{array} \right\}, \\ \text{orth} - \left\{ \begin{array}{l} \text{ori} - 1 \quad \text{siz} - 2 \quad \text{end} - 3, \\ \text{ori} - 2 \quad \text{siz} - 3 \quad \text{end} - 5 \end{array} \right\} \end{array} \right\}, 1 \right)$
See also	<code>diffn</code> , <code>two_orth_include</code> , <code>diffn_column</code> .
Key words	decomposition, geometrical constraint, positioning constraint, orthotope.

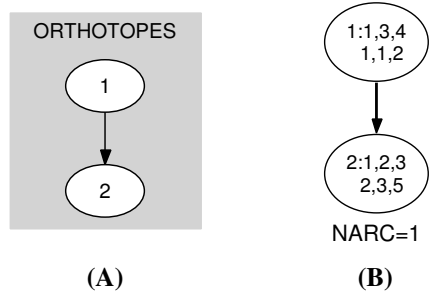


Figure 4.153: Initial and final graph of the `diffn_include` constraint

4.73 discrepancy

Origin	[103] and [104]
Constraint	discrepancy(VARIABLES, K)
Argument(s)	VARIABLES : collection(var - dvar, bad - sint) K : int
Restriction(s)	required(VARIABLES, var) required(VARIABLES, bad) $K \geq 0$ $K \leq \text{VARIABLES} $
Purpose	K is the number of variables of the collection VARIABLES which take their value in their respective sets of bad values.
Arc input(s)	VARIABLES
Arc generator	<i>SELF</i> \mapsto collection(variables)
Arc arity	1
Arc constraint(s)	in_set(variables.var, variables.bad)
Graph property(ies)	NARC = K

Example

$$\text{discrepancy} \left(\left(\begin{array}{ll} \text{var} - 4 & \text{bad} - \{1, 4, 6\}, \\ \text{var} - 5 & \text{bad} - \{0, 1\}, \\ \text{var} - 5 & \text{bad} - \{1, 6, 9\}, \\ \text{var} - 4 & \text{bad} - \{1, 4\}, \\ \text{var} - 1 & \text{bad} - \emptyset \end{array} \right), 2 \right)$$

Parts (A) and (B) of Figure 4.154 respectively show the initial and final graph. Since we use the **NARC** graph property, the unary arcs of the final graph are stressed in bold.

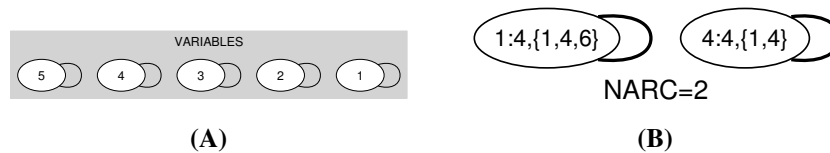


Figure 4.154: Initial and final graph of the discrepancy constraint

Graph model The arc constraint corresponds to the constraint `in_set(variables.var, variables.bad)` defined in this catalog. We employ the *SELF* arc generator in order to produce an initial graph with a single loop on each vertex.

- Remark** Limited discrepancy search was first introduced by M. L. Ginsberg and W. D. Harvey as a search technique in [105]. Later on, discrepancy based filtering was presented in the PhD thesis of F. Focacci [103, pages 171–172]. Finally the discrepancy constraint was explicitly defined in the PhD thesis of W.-J. van Hoeve [104, page 104].
- See also** among.
- Key words** value constraint, counting constraint, heuristics, limited discrepancy search.

4.74 disjoint

Origin	Derived from alldifferent.
Constraint	<code>disjoint(VARIABLES1, VARIABLES2)</code>
Argument(s)	VARIABLES1 : collection(var - dvar) VARIABLES2 : collection(var - dvar)
Restriction(s)	required(VARIABLES1, var) required(VARIABLES2, var)
Purpose	Each variable of the collection VARIABLES1 should take a value that is distinct from all the values assigned to the variables of the collection VARIABLES2.
<hr/>	
Arc input(s)	VARIABLES1 VARIABLES2
Arc generator	<i>PRODUCT</i> \mapsto collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	variables1.var = variables2.var
Graph property(ies)	NARC = 0
<hr/>	
Example	$\text{disjoint} \left(\begin{array}{l} \{ \text{var} - 1, \text{var} - 9, \text{var} - 1, \text{var} - 5 \}, \\ \left\{ \begin{array}{l} \text{var} - 2, \\ \text{var} - 7, \\ \text{var} - 7, \\ \text{var} - 0, \\ \text{var} - 6, \\ \text{var} - 8 \end{array} \right\} \end{array} \right)$ <p>In this example, values 1, 5, 9 are used by the variables of VARIABLES1 and values 0, 2, 6, 7, 8 by the variables of VARIABLES2. Since there is no intersection between the two previous sets of values the <code>disjoint</code> constraint holds. Figure 4.155 shows the initial graph. Since we use the NARC = 0 graph property the final graph is empty.</p>
Graph model	<i>PRODUCT</i> is used in order to generate the arcs of the graph between all variables of VARIABLES1 and all variables of VARIABLES2. Since we use the graph property NARC = 0 the final graph will be empty.
Signature	Since 0 is the smallest number of arcs of the final graph we can rewrite NARC = 0 to NARC \leq 0 . This leads to simplify <u>NARC</u> to NARC .
Automaton	Figure 4.156 depicts the automaton associated to the <code>disjoint</code> constraint. To each variable VAR1 _{<i>i</i>} of the collection VARIABLES1 corresponds a signature variable S _{<i>i</i>} , which is equal to 0. To each variable VAR2 _{<i>i</i>} of the collection VARIABLES2 corresponds a signature variable S _{<i>i</i>+ VARIABLES1} , which is equal to 1.

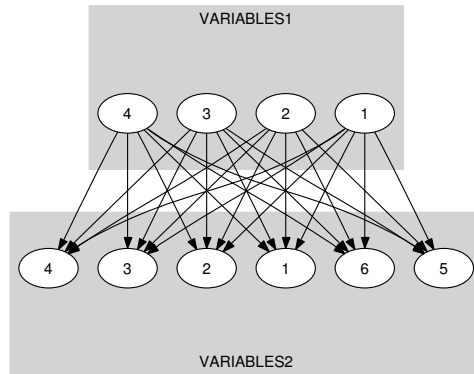


Figure 4.155: Initial graph of the disjoint constraint (the final graph is empty)

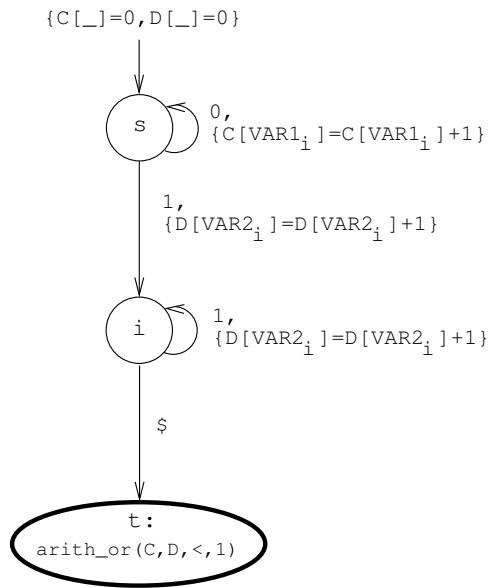


Figure 4.156: Automaton of the disjoint constraint

- Remark** Despite the fact that this is not an uncommon constraint, it can not be modelled in a compact way neither with a *disequality* constraint (i.e. two given variables have to take distinct values) nor with the `alldifferent` constraint. The `disjoint` constraint can be seen as a special case of the `common(NCOMMON1, NCOMMON2, VARIABLES1, VARIABLES2)` constraint where `NCOMMON1` and `NCOMMON2` are both set to 0.
- Algorithm** Let us note:
- n_1 the minimum number of distinct values taken by the variables of the collection `VARIABLES1`.
 - n_2 the minimum number of distinct values taken by the variables of the collection `VARIABLES2`.
 - n_{12} the maximum number of distinct values taken by the union of the variables of `VARIABLES1` and `VARIABLES2`.
- One invariant to maintain for the `disjoint` constraint is $n_1 + n_2 \leq n_{12}$. A lower bound of n_1 and n_2 can be obtained by using the algorithms provided in [33, 106]. An exact upper bound of n_{12} can be computed by using a bipartite matching algorithm.
- See also** `disjoint_tasks`.
- Key words** value constraint, empty intersection, disequality, bipartite matching, automaton, automaton with array of counters.

20000315

439

4.75 disjoint_tasks

Origin	Derived from disjoint.
Constraint	disjoint_tasks(TASKS1, TASKS2)
Argument(s)	TASKS1 : collection(origin - dvar, duration - dvar, end - dvar) TASKS2 : collection(origin - dvar, duration - dvar, end - dvar)
Restriction(s)	require_at_least(2, TASKS1, [origin, duration, end]) TASKS1.duration ≥ 0 require_at_least(2, TASKS2, [origin, duration, end]) TASKS2.duration ≥ 0
Purpose	Each task of the collection TASKS1 should not overlap any task of the collection TASKS2.
<hr/>	
Arc input(s)	TASKS1
Arc generator	<i>SELF</i> ↦ collection(tasks1)
Arc arity	1
Arc constraint(s)	tasks1.origin + tasks1.duration = tasks1.end
Graph property(ies)	$\text{NARC} = \text{TASKS1} $
<hr/>	
Arc input(s)	TASKS2
Arc generator	<i>SELF</i> ↦ collection(tasks2)
Arc arity	1
Arc constraint(s)	tasks2.origin + tasks2.duration = tasks2.end
Graph property(ies)	$\text{NARC} = \text{TASKS2} $
<hr/>	
Arc input(s)	TASKS1 TASKS2
Arc generator	<i>PRODUCT</i> ↦ collection(tasks1, tasks2)
Arc arity	2
Arc constraint(s)	<ul style="list-style-type: none"> • tasks1.duration > 0 • tasks2.duration > 0 • tasks1.origin < tasks2.end • tasks2.origin < tasks1.end
Graph property(ies)	$\text{NARC} = 0$

Example

$$\text{disjoint_tasks} \left(\left\{ \begin{array}{l} \text{origin} - 6 \quad \text{duration} - 5 \quad \text{end} - 11, \\ \text{origin} - 8 \quad \text{duration} - 2 \quad \text{end} - 10 \\ \text{origin} - 2 \quad \text{duration} - 2 \quad \text{end} - 4, \\ \text{origin} - 3 \quad \text{duration} - 3 \quad \text{end} - 6, \\ \text{origin} - 12 \quad \text{duration} - 1 \quad \text{end} - 13 \end{array} \right\} \right)$$

Figure 4.157 shows the initial graph of the third graph constraint. Because of the graph property $\text{NARC} = 0$ the corresponding final graph is empty. Figure 4.158 displays the two groups of tasks (i.e. the tasks of TASKS1 and the tasks of TASKS2). Since no task of the first group overlaps any task of the second group, the `disjoint_tasks` constraint holds.

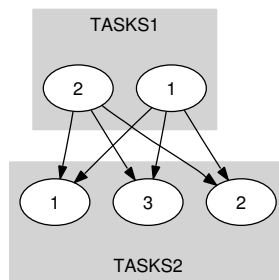


Figure 4.157: Initial graph of the `disjoint_tasks` constraint (the final graph is empty)

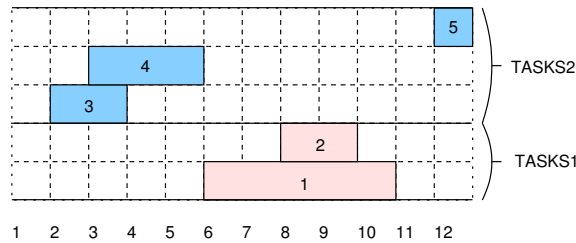


Figure 4.158: Fixed tasks of the `disjoint_tasks` constraint

Graph model

PRODUCT is used in order to generate the arcs of the graph between all the tasks of the collection `TASKS1` and all tasks of the collection `TASKS2`.

The first two graph constraints respectively enforce for each task of `TASKS1` and `TASKS2` the fact that the end of a task is equal to the sum of its origin and its duration.

The arc constraint of the third graph constraint depicts the fact that two tasks overlap. Therefore, since we use the graph property $\text{NARC} = 0$ the final graph associated to the third graph constraint will be empty and no task of `TASKS1` will overlap any task of `TASKS2`.

Signature

Since `TASKS1` is the maximum number of arcs of the final graph associated to the first graph constraint we can rewrite $\text{NARC} = |\text{TASKS1}|$. This leads to simplify NARC to NARC .

We can apply a similar remark for the second graph constraint.

Finally, since 0 is the smallest number of arcs of the final graph we can rewrite $\text{NARC} = 0$ to $\text{NARC} \leq 0$. This leads to simplify $\overline{\text{NARC}}$ to $\underline{\text{NARC}}$.

Remark

Despite the fact that this is not an uncommon constraint, it cannot be modelled in a compact way with one single `cumulative` constraint. But it can be expressed by using the `coloured_cumulative` constraint: We assign a first colour to the tasks of `TASKS1` as well as a second distinct colour to the tasks of `TASKS2`. Finally we set up a limit of 1 for the maximum number of distinct colours allowed at each time point.

See also

`disjoint`, `coloured_cumulative`.

Key words

scheduling constraint, temporal constraint, non-overlapping.

20030820

443

4.76 disjunctive

Origin	[107]
Constraint	<code>disjunctive(TASKS)</code>
Synonym(s)	<code>one_machine.</code>
Argument(s)	<code>TASKS : collection(origin - dvar, duration - dvar)</code>
Restriction(s)	<code>required(TASKS, [origin, duration])</code> <code>TASKS.duration ≥ 0</code>
Purpose	All the tasks of the collection TASKS should not overlap.
<hr/>	
Arc input(s)	<code>TASKS</code>
Arc generator	$\text{CLIQUE}(<) \mapsto \text{collection}(\text{tasks1}, \text{tasks2})$
Arc arity	2
Arc constraint(s)	$\bigvee \left(\begin{array}{l} \text{tasks1.duration} = 0, \\ \text{tasks2.duration} = 0, \\ \text{tasks1.origin} + \text{tasks1.duration} \leq \text{tasks2.origin}, \\ \text{tasks2.origin} + \text{tasks2.duration} \leq \text{tasks1.origin} \end{array} \right)$
Graph property(ies)	$\text{NARC} = \text{TASKS} * (\text{TASKS} - 1)/2$
<hr/>	
Example	$\text{disjunctive} \left(\left(\begin{array}{l} \left(\begin{array}{ll} \text{origin} - 1 & \text{duration} - 3, \\ \text{origin} - 2 & \text{duration} - 0, \end{array} \right) \\ \left(\begin{array}{ll} \text{origin} - 7 & \text{duration} - 2, \\ \text{origin} - 4 & \text{duration} - 1 \end{array} \right) \end{array} \right) \right)$ <p>Parts (A) and (B) of Figure 4.159 respectively show the initial and final graph. The <code>disjunctive</code> constraint holds since all the arcs of the initial graph belong to the final graph: all the non-overlapping constraints holds.</p>
Graph model	We generate a <i>clique</i> with a non-overlapping constraint between each pair of distinct tasks and state that the number of arcs of the final graph should be equal to the number of arcs of the initial graph.
Remark	A soft version of this constraint, under the hypothesis that all durations are fixed, was presented by P. Baptiste et al. in [108]. In this context the goal was to perform as many tasks as possible within their respective due-dates.
Algorithm	Efficient filtering algorithms for handling the <code>disjunctive</code> constraint are described in [109] and [110].
See also	<code>cumulative</code> , <code>diffn</code> .
Key words	scheduling constraint, resource constraint, decomposition.

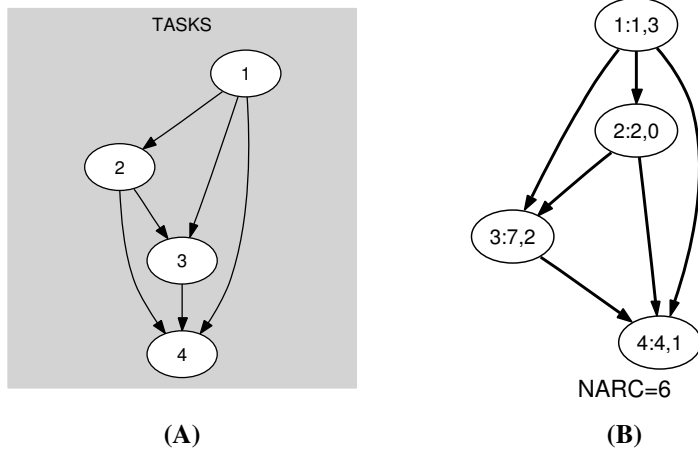


Figure 4.159: Initial and final graph of the disjunctive constraint

4.77 distance_between

Origin	N. Beldiceanu
Constraint	distance_between(DIST, VARIABLES1, VARIABLES2, CTR)
Argument(s)	DIST : dvar VARIABLES1 : collection(var - dvar) VARIABLES2 : collection(var - dvar) CTR : atom
Restriction(s)	DIST ≥ 0 DIST $\leq \text{VARIABLES1} * \text{VARIABLES2} $ required(VARIABLES1, var) required(VARIABLES2, var) $ \text{VARIABLES1} = \text{VARIABLES2} $ CTR $\in [=, \neq, <, \geq, >, \leq]$
Purpose	<div style="border: 1px solid black; padding: 5px;"> <p>Let U_i and V_i be respectively the i^{th} and j^{th} variables ($i \neq j$) of the collection VARIABLES1. In a similar way, let X_i and Y_i be respectively the i^{th} and j^{th} variables ($i \neq j$) of the collection VARIABLES2. DIST is equal to the number of times one of the following mutually incompatible conditions are true:</p> <ul style="list-style-type: none"> • U_i CTR V_i holds and X_i CTR Y_i does not hold, • X_i CTR Y_i holds and U_i CTR V_i does not hold. </div>
Arc input(s)	VARIABLES1/ VARIABLES2
Arc generator	<i>CLIQUE</i> (\neq) \mapsto collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	variables1.var CTR variables2.var
Graph property(ies)	DISTANCE = DIST

Example	$\text{distance_between} \left(2, \left\{ \begin{array}{l} \text{var} - 3, \\ \text{var} - 4, \\ \text{var} - 6, \\ \text{var} - 2, \\ \text{var} - 4 \end{array} \right\}, \left\{ \begin{array}{l} \text{var} - 2, \\ \text{var} - 6, \\ \text{var} - 9, \\ \text{var} - 3, \\ \text{var} - 6 \end{array} \right\}, < \right)$
----------------	--

Between solution var-3,var-4,var-6,var-2,var-4 and solution var-2,var-6,var-9,var-3,var-6 there are 2 changes, which respectively correspond to:

- Within the final graph associated to solution `var-3,var-4,var-6,var-2,var-4` the arc $4 \rightarrow 1$ (i.e. values $2 \rightarrow 3$) does not occur in the final graph associated to `var-2,var-6,var-9,var-3,var-6`,
- Within the final graph associated to solution `var-2,var-6,var-9,var-3,var-6` the arc $1 \rightarrow 4$ (i.e. values $2 \rightarrow 3$) does not occur in the final graph associated to `var-3,var-4,var-6,var-2,var-4`.

Part (A) of Figure 4.160 gives the final graph associated to the solution `var-3,var-4,var-6,var-2,var-4`, while part (B) shows the final graph corresponding to `var-2,var-6,var-9,var-3,var-6`. The two arc constraints that differ from one graph to the other are marked by a dotted line.

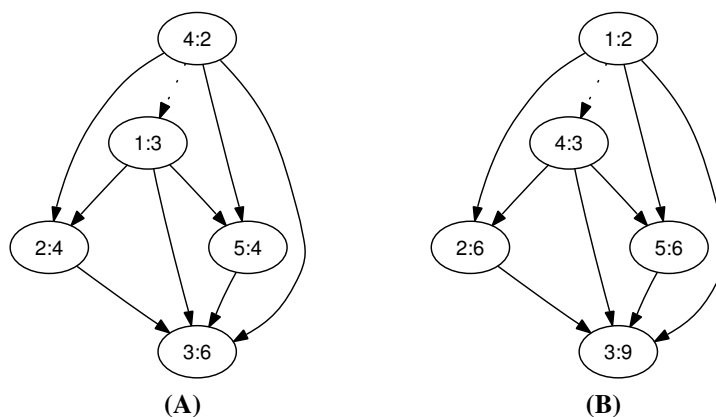


Figure 4.160: Final graphs of the `distance_between` constraint

Graph model

Within the arc input field, the character `/` indicates that we generate two distinct graphs. The graph property **DISTANCE** measures the distance between two digraphs G_1 and G_2 . This distance is defined as the sum of the following quantities:

- The number of arcs of G_1 which do not belong to G_2 ,
- The number of arcs of G_2 which do not belong to G_1 .

Usage

Measure the distance between two solutions in term of the number of constraint changes. This should be put in contrast to the number of value changes which is sometimes superficial.

See also

`distance_change`.

Key words

proximity constraint.

4.78 distance_change

Origin	Derived from change.
Constraint	distance_change(DIST, VARIABLES1, VARIABLES2, CTR)
Argument(s)	DIST : dvar VARIABLES1 : collection(var - dvar) VARIABLES2 : collection(var - dvar) CTR : atom
Restriction(s)	DIST \geq 0 DIST < VARIABLES1 required(VARIABLES1, var) required(VARIABLES2, var) VARIABLES1 = VARIABLES2 CTR \in [=, \neq , <, \geq , >, \leq]
Purpose	DIST is equal to the number of times one of the following two conditions is true ($1 \leq i < n$): <ul style="list-style-type: none"> • VARIABLES1[i].var CTR VARIABLES1[i + 1].var holds and VARIABLES2[i].var CTR VARIABLES2[i + 1].var does not hold, • VARIABLES2[i].var CTR VARIABLES2[i + 1].var holds and VARIABLES1[i].var CTR VARIABLES1[i + 1].var does not hold.
Arc input(s)	VARIABLES1/ VARIABLES2
Arc generator	<i>PATH</i> \mapsto collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	variables1.var CTR variables2.var
Graph property(ies)	DISTANCE = DIST

Example	$\text{distance_change} \left(1, \left\{ \begin{array}{l} \text{var} - 3, \\ \text{var} - 3, \\ \text{var} - 1, \\ \text{var} - 2, \\ \text{var} - 2 \end{array} \right\}, \left\{ \begin{array}{l} \text{var} - 4, \\ \text{var} - 4, \\ \text{var} - 3, \\ \text{var} - 3, \\ \text{var} - 3 \end{array} \right\}, \neq \right)$
----------------	--

Part (A) of Figure 4.161 gives the final graph associated to the solution var-3,var-3,var-1,var-2,var-2, while part (B) shows the final graph corresponding to var-4,var-4,var-3,var-3,var-3. Since arc $3 \rightarrow 4$ belongs to the first final graph but not to the second one, the distance between the two final graphs is equal to 1.

Graph model	<p>Within the arc input field, the character / indicates that we generate two distinct graphs. The graph property DISTANCE measures the distance between two digraphs G_1 and G_2. This distance is defined as the sum of the following quantities:</p> <ul style="list-style-type: none"> • The number of arcs of G_1 which do not belong to G_2, • The number of arcs of G_2 which do not belong to G_1.
Automaton	<p>Figure 4.162 depicts the automaton associated to the distance_change constraint. Let $(VAR1_i, VAR1_{i+1})$ and $(VAR2_i, VAR2_{i+1})$ respectively be the i^{th} pairs of consecutive variables of the collections VARIABLES1 and VARIABLES2. To each quadruple $(VAR1_i, VAR1_{i+1}, VAR2_i, VAR2_{i+1})$ corresponds a 0-1 signature variable S_i. The following signature constraint links these variables:</p> $((VAR1_i = VAR1_{i+1}) \wedge (VAR2_i \neq VAR2_{i+1})) \vee ((VAR1_i \neq VAR1_{i+1}) \wedge (VAR2_i = VAR2_{i+1})) \Leftrightarrow S_i.$
Usage	Measure the distance between two solutions according to the change constraint.
Remark	We measure that distance according to a given constraint and not according to the fact that the variables take distinct values.
See also	change, distance_between.
Key words	proximity constraint, automaton, automaton with counters, sliding cyclic(2) constraint network(2).

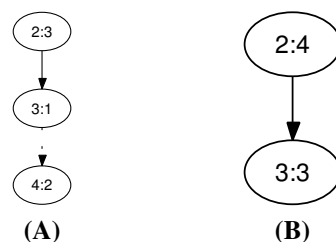


Figure 4.161: Final graphs of the distance_change constraint

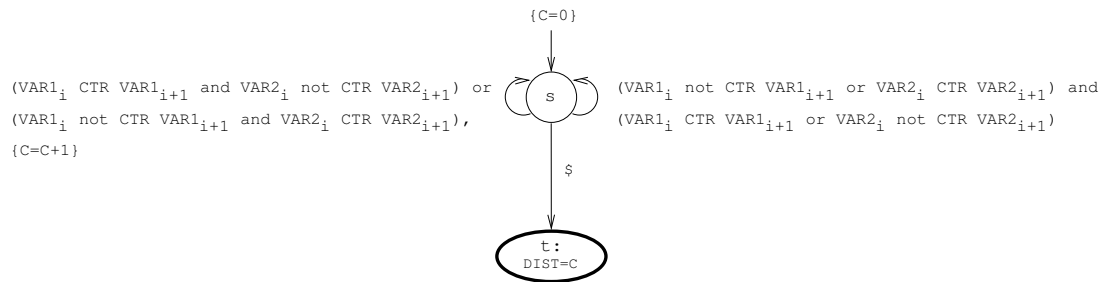


Figure 4.162: Automaton of the distance_change constraint

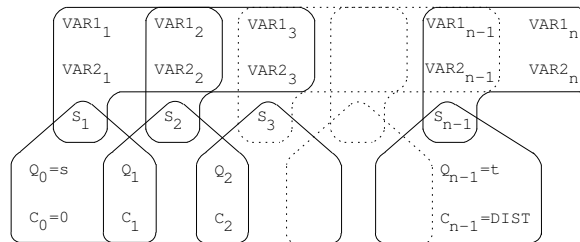


Figure 4.163: Hypergraph of the reformulation corresponding to the automaton of the distance_change constraint

20000128

451

4.79 domain_constraint

Origin	[111]
Constraint	domain_constraint(VAR, VALUES)
Argument(s)	VAR : dvar VALUES : collection(var01 – dvar, value – int)
Restriction(s)	required(VALUES, [var01, value]) VALUES.var01 ≥ 0 VALUES.var01 ≤ 1 distinct(VALUES, value)
Purpose	Make the link between a domain variable VAR and those 0-1 variables that are associated to each potential value of VAR: The 0-1 variable associated to the value which is taken by variable VAR is equal to 1, while the remaining 0-1 variables are all equal to 0.
Derived Collection(s)	<u>col(VALUE – collection(var01 – int, value – dvar), [item(var01 – 1, value – VAR)])</u>
Arc input(s)	VALUE VALUES
Arc generator	<i>PRODUCT</i> ↦ collection(value, values)
Arc arity	2
Arc constraint(s)	value.value = values.value ⇔ values.var01 = 1
Graph property(ies)	<u>NARC = VALUES </u>

Example domain_constraint $\left(5, \left\{ \begin{array}{ll} \text{var01} - 0 & \text{value} - 9, \\ \text{var01} - 1 & \text{value} - 5, \\ \text{var01} - 0 & \text{value} - 2, \\ \text{var01} - 0 & \text{value} - 7 \end{array} \right\} \right)$

In the previous example, the 0-1 variable associated to value 5 is set to 1, while the other 0-1 variables are all set to 0. Parts (A) and (B) of Figure 4.164 respectively show the initial and final graph. Since we use the **NARC** graph property, the arcs of the final graph are stressed in bold.

Graph model The domain_constraint constraint is modelled with the following bipartite graph:

- The first class of vertices corresponds to one single vertex containing the domain variable.
- The second class of vertices contains one vertex for each item of the collection VALUES.

PRODUCT is used in order to generate the arcs of the graph. In our context it takes a collection with one single item {var01 – 1 value – VAR} and the collection *VALUES*.

The arc constraint between the variable *VAR* and one potential value v expresses the following:

- If the 0-1 variable associated to v is equal to 1, *VAR* is equal to v .
- Otherwise, if the 0-1 variable associated to v is equal to 0, *VAR* is not equal to v .

Since all arc constraints should hold the final graph contains exactly $|VALUES|$ arcs.

Signature	Since the number of arcs of the initial graph is equal to <i>VALUES</i> the maximum number of arcs of the final graph is also equal to <i>VALUES</i> . Therefore we can rewrite the graph property $\overline{NARC} = VALUES $ to $NARC \geq VALUES $. This leads to simplify \overline{NARC} to \overline{NARC} .
Automaton	Figure 4.165 depicts the automaton associated to the <i>domain_constraint</i> constraint. Let $VAR01_i$ and $VALUE_i$ respectively be the <i>var01</i> and the <i>value</i> attributes of the i^{th} item of the <i>VALUES</i> collection. To each triple $(VAR, VAR01_i, VALUE_i)$ corresponds a 0-1 signature variable S_i as well as the following signature constraint: $((VAR = VALUE_i) \Leftrightarrow VAR01_i) \Leftrightarrow S_i$.
Usage	This constraint is used in order to make the link between a formulation using finite domain constraints and a formulation exploiting 0-1 variables.
See also	<code>link_set_to_booleans</code> .
Key words	decomposition, channeling constraint, domain channel, boolean channel, linear programming, automaton, automaton without counters, centered cyclic(1) constraint network(1), derived collection.

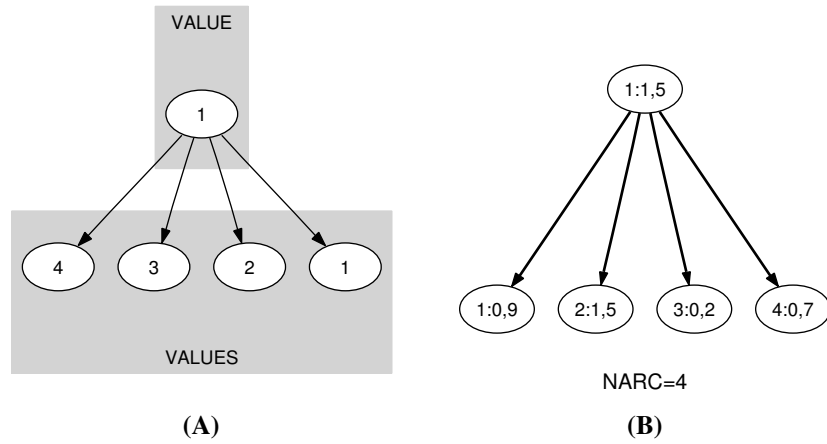


Figure 4.164: Initial and final graph of the domain_constraint constraint

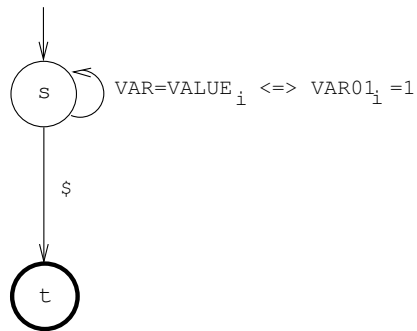


Figure 4.165: Automaton of the domain_constraint constraint

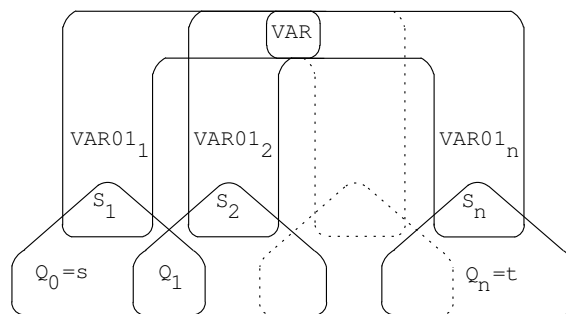


Figure 4.166: Hypergraph of the reformulation corresponding to the automaton of the domain_constraint constraint

20030820

455

4.80 elem

Origin	Derived from <code>element</code> .
Constraint	<code>elem(ITEM, TABLE)</code>
Usual name	<code>element</code>
Argument(s)	<p><code>ITEM</code> : <code>collection(index - dvar, value - dvar)</code> <code>TABLE</code> : <code>collection(index - int, value - dvar)</code></p>
Restriction(s)	<p><code>required(ITEM, [index, value])</code> <code>ITEM.index ≥ 1</code> <code>ITEM.index ≤ TABLE </code> <code> ITEM = 1</code> <code>required(TABLE, [index, value])</code> <code>TABLE.index ≥ 1</code> <code>TABLE.index ≤ TABLE </code> <code>distinct(TABLE, index)</code></p>
Purpose	ITEM is equal to one of the entries of the table TABLE.
<hr/>	
Arc input(s)	ITEM TABLE
Arc generator	<i>PRODUCT</i> \mapsto <code>collection(item, table)</code>
Arc arity	2
Arc constraint(s)	<ul style="list-style-type: none"> • <code>item.index = table.index</code> • <code>item.value = table.value</code>
Graph property(ies)	NARC = 1
<hr/>	
Example	$\text{elem} \left(\left(\begin{array}{l} \{ \text{index} - 3 \text{ value} - 2 \}, \\ \left\{ \begin{array}{l} \text{index} - 1 \text{ value} - 6, \\ \text{index} - 2 \text{ value} - 9, \\ \text{index} - 3 \text{ value} - 2, \\ \text{index} - 4 \text{ value} - 9 \end{array} \right\} \end{array} \right) \right)$
	Parts (A) and (B) of Figure 4.167 respectively show the initial and final graph. Since we use the NARC graph property, the unique arc of the final graph is stressed in bold.
Graph model	We regroup the <code>INDEX</code> and <code>VALUE</code> parameters of the original <code>element</code> constraint <code>element(INDEX, TABLE, VALUE)</code> into the parameter <code>ITEM</code> . We also make explicit the different indices of the table <code>TABLE</code> .
Signature	Since all the <code>index</code> attributes of <code>TABLE</code> are distinct and because of the first condition of the arc constraint the final graph cannot have more than one arc. Therefore we can rewrite NARC = 1 to NARC ≥ 1 and simplify <u>NARC</u> to <u>NARC</u> .

Automaton

Figure 4.168 depicts the automaton associated to the `elem` constraint. Let `INDEX` and `VALUE` respectively be the `index` and the `value` attributes of the unique item of the `ITEM` collection. Let `INDEXi` and `VALUEi` respectively be the `index` and the `value` attributes of the i^{th} item of the `TABLE` collection. To each quadruple $(INDEX, VALUE, INDEX_i, VALUE_i)$ corresponds a 0-1 signature variable S_i as well as the following signature constraint: $((INDEX = INDEX_i) \wedge (VALUE = VALUE_i)) \Leftrightarrow S_i$.

Usage

Makes the link between the decision variable `INDEX` and the variable `VALUE` according to a given table of values `TABLE`. We now give three typical uses of the `elem` constraint.

1. In some scheduling problems the duration of a task depends on the machine where the task will be assigned in final schedule. In this case we generate for each task an `elem` constraint of the following form:

$$\text{elem} \left(\left(\left\{ \begin{array}{cc} \text{index} - \text{Machine} & \text{value} - \text{Duration} \end{array} \right\}, \right. \right. \\ \left. \left. \left(\begin{array}{cc} \text{index} - 1 & \text{value} - \text{Dur}_1, \\ \text{index} - 2 & \text{value} - \text{Dur}_2, \\ & \vdots \\ \text{index} - m & \text{value} - \text{Dur}_m \end{array} \right) \right) \right)$$

where:

- `Machine` is a domain variable which indicates the resource to which the task will be assigned,
 - `Duration` is a domain variable which corresponds to the duration of the task,
 - `Dur1, Dur2, ..., Durm` are the respective durations of the task according to the hypothesis that it runs on machine 1, 2 or m .
2. In some vehicle routing problems we typically use the `elem` constraint to express the distance between the i^{th} location and the next location visited by a vehicle. For this purpose we generate for each location i an `elem` constraint of the form:

$$\text{elem} \left(\left(\left\{ \begin{array}{cc} \text{index} - \text{Next}_i & \text{value} - \text{distance}_i \end{array} \right\}, \right. \right. \\ \left. \left. \left(\begin{array}{cc} \text{index} - 1 & \text{value} - \text{Dist}_{i_1}, \\ \text{index} - 2 & \text{value} - \text{Dist}_{i_2}, \\ & \vdots \\ \text{index} - m & \text{value} - \text{Dist}_{i_m} \end{array} \right) \right) \right)$$

where:

- `Nexti` is a domain variable which gives the index of the location the vehicle will visit just after the i^{th} location,
 - `distancei` is a domain variable which corresponds to the distance between location i and the location the vehicle will visit just after,
 - `Disti1, Disti2, ..., Distim` are the respective distances between location i and locations 1, 2, ..., m .
3. In some optimization problems a classical use of the `elem` constraint consists expressing the link between a discrete choice and its corresponding cost. For each discrete choice we create an `elem` constraint of the form:

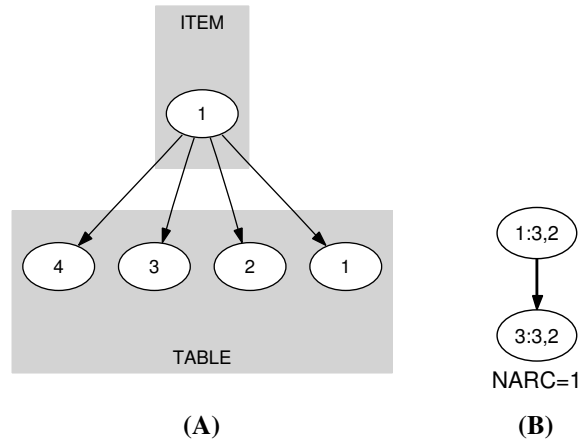


Figure 4.167: Initial and final graph of the elem constraint

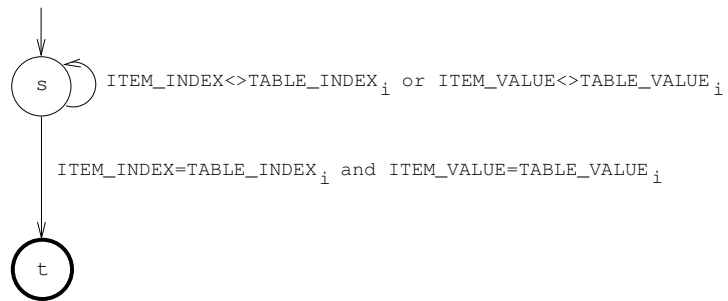


Figure 4.168: Automaton of the elem constraint

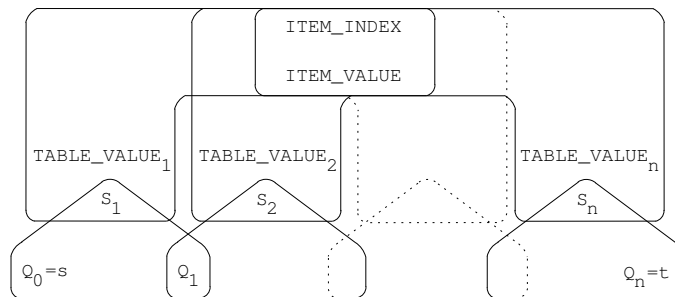


Figure 4.169: Hypergraph of the reformulation corresponding to the automaton of the elem constraint

$$\text{elem} \left(\left\{ \begin{array}{l} \text{index} - \text{Choice} \quad \text{value} - \text{Cost} \\ \left\{ \begin{array}{l} \text{index} - 1 \quad \text{value} - \text{Cost}_1, \\ \text{index} - 2 \quad \text{value} - \text{Cost}_2, \\ \vdots \\ \text{index} - m \quad \text{value} - \text{Cost}_m \end{array} \right\} \end{array} \right\}, \right)$$

where:

- **Choice** is a domain variable which indicates which alternative will be finally selected,
- **Cost** is a domain variable which corresponds to the cost of the decision associated to the value of the **Choice** variable,
- **Cost₁, Cost₂, ..., Cost_m** are the respective costs associated to the alternatives 1, 2, ..., *m*.

Remark Originally, the parameters of the `elem` constraint had the form `element(INDEX, TABLE, VALUE)`, where `INDEX` and `VALUE` were two domain variables and `TABLE` a list of non-negative integers.

See also `element`, `element_greatereq`, `element_lesseq`, `element_sparse`, `element_matrix`, `elements`, `elements_alldifferent`, `stage_element`.

Key words array constraint, data constraint, table, functional dependency, variable indexing, variable subscript, automaton, automaton without counters, centered cyclic(2) constraint network(1).

4.81 element

Origin	[32]
Constraint	element(INDEX, TABLE, VALUE)
Argument(s)	INDEX : dvar TABLE : collection(value - dvar) VALUE : dvar
Restriction(s)	INDEX \geq 1 INDEX \leq TABLE required(TABLE, value)
Purpose	VALUE is equal to the INDEX th item of TABLE.
Derived Collection(s)	$\text{col} \left(\begin{array}{l} \text{ITEM} - \text{collection}(\text{index} - \text{dvar}, \text{value} - \text{dvar}), \\ [\text{item}(\text{index} - \text{INDEX}, \text{value} - \text{VALUE})] \end{array} \right)$
Arc input(s)	ITEM TABLE
Arc generator	PRODUCT \mapsto collection(item, table)
Arc arity	2
Arc constraint(s)	<ul style="list-style-type: none"> • item.index = table.key • item.value = table.value
Graph property(ies)	NARC = 1
Example	$\text{element} \left(3, \left\{ \begin{array}{l} \text{value} - 6, \\ \text{value} - 9, \\ \text{value} - 2, \\ \text{value} - 9 \end{array} \right\}, 2 \right)$ <p>Parts (A) and (B) of Figure 4.170 respectively show the initial and final graph. Since we use the NARC graph property, the unique arc of the final graph is stressed in bold.</p>
Graph model	The original element constraint with three arguments. We use the derived collection ITEM for putting together the INDEX and VALUE parameters of the element constraint. Within the arc constraint we use the implicit attribute key which associates to each item of a collection its position within the collection.
Signature	Because of the first condition of the arc constraint the final graph cannot have more than one arc. Therefore we can rewrite NARC = 1 to NARC \geq 1 and simplify <u>NARC</u> to <u>NARC</u> .

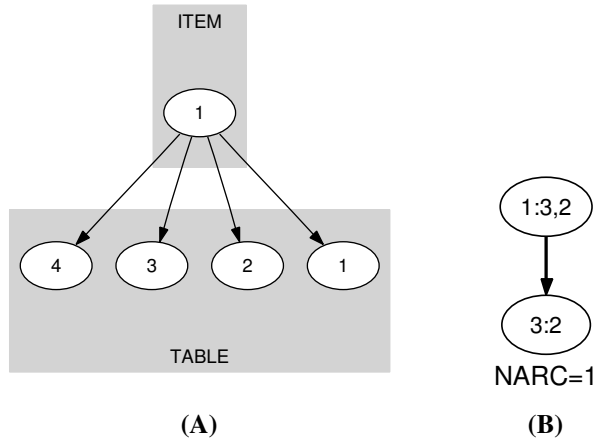


Figure 4.170: Initial and final graph of the element constraint

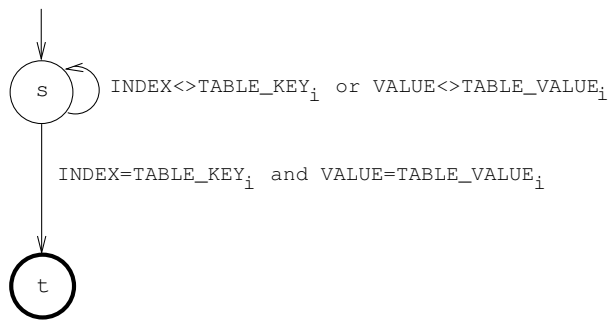


Figure 4.171: Automaton of the element constraint

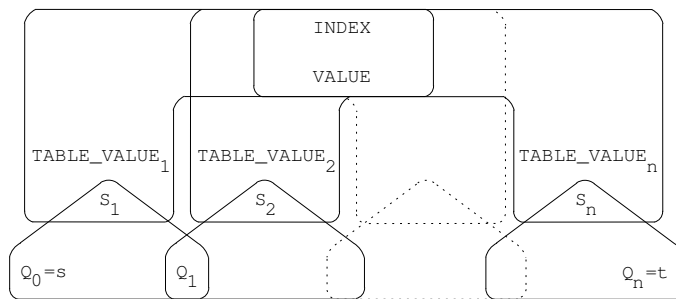


Figure 4.172: Hypergraph of the reformulation corresponding to the automaton of the element constraint

Automaton	Figure 4.171 depicts the automaton associated to the <code>element</code> constraint. Let $VALUE_i$ be the value attribute of the i^{th} item of the <code>TABLE</code> collection. To each triple $(INDEX, VALUE, VALUE_i)$ corresponds a 0-1 signature variable S_i as well as the following signature constraint: $(INDEX = i \wedge VALUE = VALUE_i) \Leftrightarrow S_i$.
Usage	See <code>elem</code> .
Remark	In the original <code>element</code> constraint of CHIP the <code>index</code> attribute was not explicitly present in the table of values. It was implicitly defined as the position of a value in the previous table. The case constraint [46] is a generalization of the <code>element</code> constraint, where the table is replaced by a directed acyclic graph describing the set of solutions.
See also	<code>elem</code> , <code>element_greatereq</code> , <code>element_lesseq</code> , <code>element_sparse</code> , <code>element_matrix</code> , <code>elements</code> , <code>elements_alldifferent</code> , <code>stage_element</code> .
Key words	array constraint, data constraint, table, functional dependency, variable indexing, variable subscript, automaton, automaton without counters, centered cyclic(2) constraint network(1), derived collection.

20000128

463

4.82 element_greatereq

Origin	[112]
Constraint	element_greatereq(ITEM, TABLE)
Argument(s)	ITEM : collection(index – dvar, value – dvar) TABLE : collection(index – int, value – int)
Restriction(s)	required(ITEM, [index, value]) ITEM.index ≥ 1 ITEM.index ≤ TABLE ITEM = 1 required(TABLE, [index, value]) TABLE.index ≥ 1 TABLE.index ≤ TABLE distinct(TABLE, index)
Purpose	ITEM.value is greater than or equal to one of the entries (i.e. the value attribute) of the table TABLE.
<hr/>	
Arc input(s)	ITEM TABLE
Arc generator	PRODUCT ↦ collection(item, table)
Arc arity	2
Arc constraint(s)	<ul style="list-style-type: none"> • item.index = table.index • item.value ≥ table.value
Graph property(ies)	NARC = 1
<hr/>	
Example	$\text{element_greatereq} \left(\left(\begin{array}{l} \{ \text{index} - 1 \text{ value} - 8 \}, \\ \left\{ \begin{array}{ll} \text{index} - 1 & \text{value} - 6, \\ \text{index} - 2 & \text{value} - 9, \\ \text{index} - 3 & \text{value} - 2, \\ \text{index} - 4 & \text{value} - 9 \end{array} \right\} \end{array} \right) \right)$
	Parts (A) and (B) of Figure 4.173 respectively show the initial and final graph. Since we use the NARC graph property, the unique arc of the final graph is stressed in bold.
Graph model	Similar to the element constraint except that the <i>equality</i> constraint of the second condition of the arc constraint is replaced by a <i>greater than or equal to</i> constraint.
Signature	Since all the index attributes of TABLE are distinct and because of the first arc constraint the final graph cannot have more than one arc. Therefore we can rewrite NARC = 1 to NARC ≥ 1 and simplify <u>NARC</u> to <u>NARC</u> .

- Automaton** Figure 4.174 depicts the automaton associated to the `element_greatereq` constraint. Let `INDEX` and `VALUE` respectively be the `index` and the `value` attributes of the unique item of the `ITEM` collection. Let `INDEXi` and `VALUEi` respectively be the `index` and the `value` attributes of the i^{th} item of the `TABLE` collection. To each quadruple $(INDEX, VALUE, INDEX_i, VALUE_i)$ corresponds a 0-1 signature variable S_i as well as the following signature constraint: $((INDEX = INDEX_i) \wedge (VALUE \geq VALUE_i)) \Leftrightarrow S_i$.
- Usage** Used for modelling variable subscripts in linear constraints [112].
- See also** `element`, `element_lesseq`.
- Key words** array constraint, data constraint, binary constraint, table, linear programming, variable subscript, variable indexing, automaton, automaton without counters, centered cyclic(2) constraint network(1).

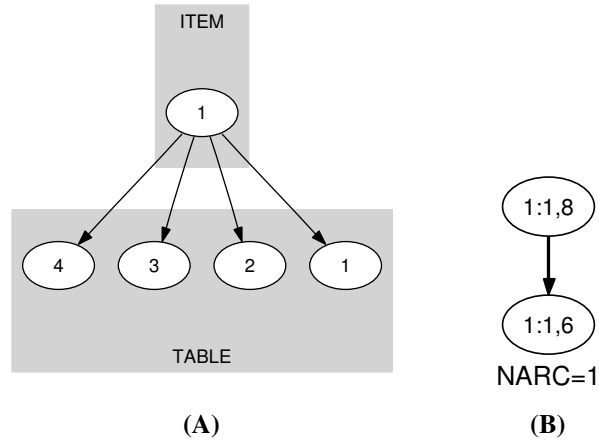


Figure 4.173: Initial and final graph of the `element_greatereq` constraint

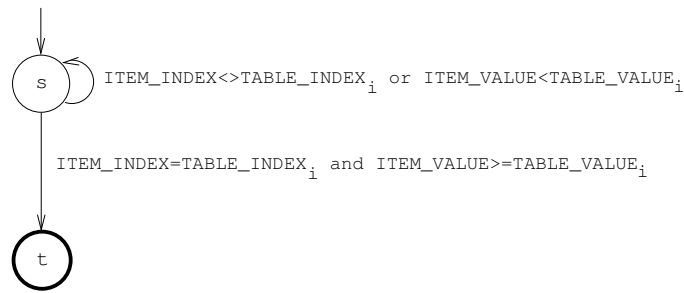


Figure 4.174: Automaton of the `element_greatereq` constraint

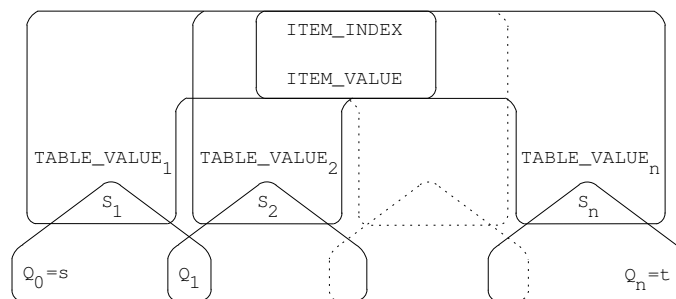


Figure 4.175: Hypergraph of the reformulation corresponding to the automaton of the `element_greatereq` constraint

20030820

467

4.83 element_lesseq

Origin	[112]
Constraint	element_lesseq(ITEM, TABLE)
Argument(s)	ITEM : collection(index - dvar, value - dvar) TABLE : collection(index - int, value - int)
Restriction(s)	required(ITEM, [index, value]) ITEM.index ≥ 1 ITEM.index ≤ TABLE ITEM = 1 required(TABLE, [index, value]) TABLE.index ≥ 1 TABLE.index ≤ TABLE distinct(TABLE, index)
Purpose	ITEM.value is less than or equal to one of the entries (i.e. the value attribute) of the table TABLE.
<hr/>	
Arc input(s)	ITEM TABLE
Arc generator	PRODUCT ↦ collection(item, table)
Arc arity	2
Arc constraint(s)	<ul style="list-style-type: none"> • item.index = table.index • item.value ≤ table.value
Graph property(ies)	NARC = 1
<hr/>	
Example	$\text{element_lesseq} \left(\left(\begin{array}{l} \{ \text{index} - 3 \text{ value} - 1 \}, \\ \left\{ \begin{array}{ll} \text{index} - 1 & \text{value} - 6, \\ \text{index} - 2 & \text{value} - 9, \\ \text{index} - 3 & \text{value} - 2, \\ \text{index} - 4 & \text{value} - 9 \end{array} \right\} \end{array} \right) \right)$ <p>Parts (A) and (B) of Figure 4.176 respectively show the initial and final graph. Since we use the NARC graph property, the unique arc of the final graph is stressed in bold.</p>
Graph model	Similar to the element constraint except that the <i>equality</i> constraint of the second condition of the arc constraint is replaced by a <i>less than or equal to</i> constraint.
Signature	Since all the index attributes of TABLE are distinct and because of the first arc constraint the final graph cannot have more than one arc. Therefore we can rewrite NARC = 1 to NARC ≥ 1 and simplify <u>NARC</u> to <u>NARC</u> .

- Automaton** Figure 4.177 depicts the automaton associated to the `element_lesseq` constraint. Let `INDEX` and `VALUE` respectively be the `index` and the `value` attributes of the unique item of the `ITEM` collection. Let $INDEX_i$ and $VALUE_i$ respectively be the `index` and the `value` attributes of the i^{th} item of the `TABLE` collection. To each quadruple $(INDEX, VALUE, INDEX_i, VALUE_i)$ corresponds a 0-1 signature variable S_i as well as the following signature constraint: $((INDEX = INDEX_i) \wedge (VALUE \leq VALUE_i)) \Leftrightarrow S_i$.
- Usage** Used for modelling variable subscripts in linear constraints [112].
- See also** `element`, `element_greatereq`.
- Key words** array constraint, data constraint, binary constraint, table, linear programming, variable subscript, variable indexing, automaton, automaton without counters, centered cyclic(2) constraint network(1).

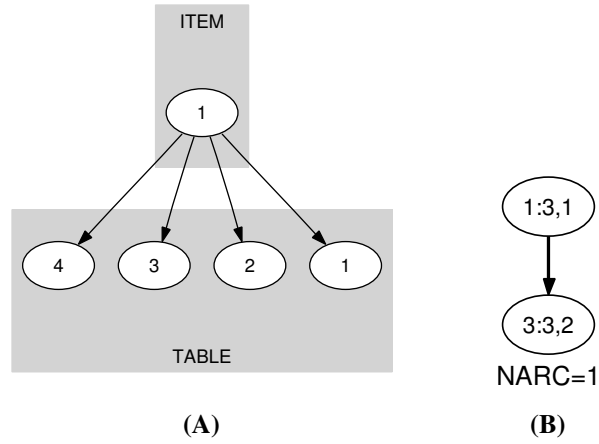


Figure 4.176: Initial and final graph of the `element_lesseq` constraint

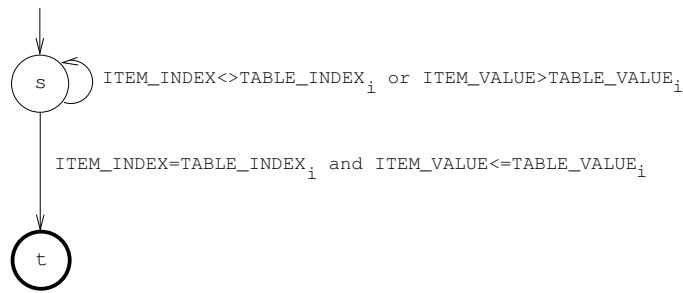


Figure 4.177: Automaton of the `element_lesseq` constraint

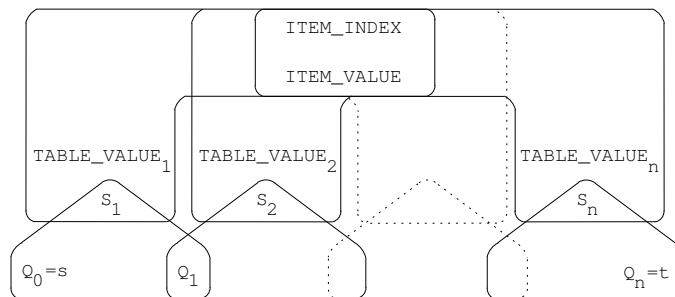


Figure 4.178: Hypergraph of the reformulation corresponding to the automaton of the `element_lesseq` constraint

4.84 element_matrix

Origin	CHIP
Constraint	element_matrix(MAX_I, MAX_J, INDEX_I, INDEX_J, MATRIX, VALUE)
Argument(s)	<pre> MAX_I : int MAX_J : int INDEX_I : dvar INDEX_J : dvar MATRIX : collection(i - int, j - int, v - int) VALUE : dvar </pre>
Restriction(s)	<pre> MAX_I ≥ 1 MAX_J ≥ 1 INDEX_I ≥ 1 INDEX_I ≤ MAX_I INDEX_J ≥ 1 INDEX_J ≤ MAX_J required(MATRIX, [i, j, v]) increasing_seq(MATRIX, [i, j]) MATRIX.i ≥ 1 MATRIX.i ≤ MAX_I MATRIX.j ≥ 1 MATRIX.j ≤ MAX_J MATRIX = MAX_I * MAX_J </pre>
Purpose	<p>The MATRIX collection corresponds to the two-dimensional matrix MATRIX[1..MAX_I, 1..MAX_J]. VALUE is equal to the entry MATRIX[INDEX_I, INDEX_J] of the previous matrix.</p>
Derived Collection(s)	$\text{col} \left(\begin{array}{l} \text{ITEM} - \text{collection}(\text{index}_i - \text{dvar}, \text{index}_j - \text{dvar}, \text{value} - \text{dvar}), \\ [\text{item}(\text{index}_i - \text{INDEX}_I, \text{index}_j - \text{INDEX}_J, \text{value} - \text{VALUE})] \end{array} \right)$
Arc input(s)	ITEM MATRIX
Arc generator	<i>PRODUCT</i> \mapsto collection(item, matrix)
Arc arity	2
Arc constraint(s)	<ul style="list-style-type: none"> • item.index_i = matrix.i • item.index_j = matrix.j • item.value = matrix.v
Graph property(ies)	NARC = 1

Example

$$\text{element_matrix} \left(4, 3, 1, 3, \left\{ \begin{array}{l} i-1 \quad j-1 \quad v-4, \\ i-1 \quad j-2 \quad v-1, \\ i-1 \quad j-3 \quad v-7, \\ i-2 \quad j-1 \quad v-1, \\ i-2 \quad j-2 \quad v-0, \\ i-2 \quad j-3 \quad v-8, \\ i-3 \quad j-1 \quad v-3, \\ i-3 \quad j-2 \quad v-2, \\ i-3 \quad j-3 \quad v-1, \\ i-4 \quad j-1 \quad v-0, \\ i-4 \quad j-2 \quad v-0, \\ i-4 \quad j-3 \quad v-6 \end{array} \right\}, 7 \right)$$

Parts (A) and (B) of Figure 4.179 respectively show the initial and final graph. Since we use the **NARC** graph property, the unique arc of the final graph is stressed in bold.

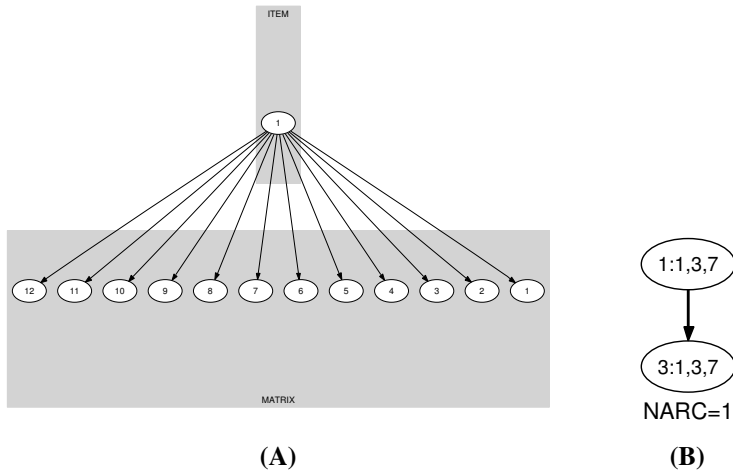


Figure 4.179: Initial and final graph of the element_matrix constraint

Graph model

Similar to the `element` constraint except that the arc constraint is updated according to the fact that we have a two-dimensional matrix.

Signature

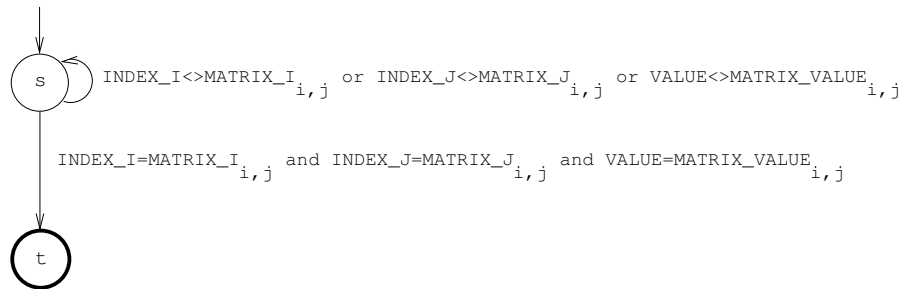
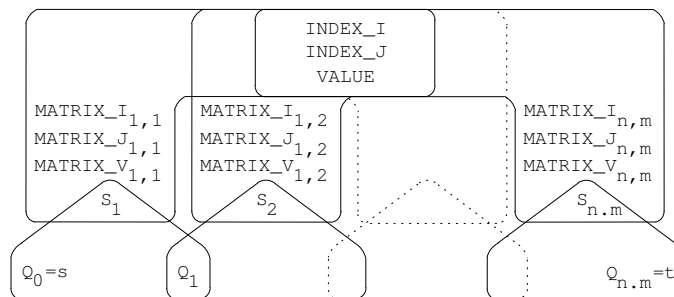
Because of the first condition of the arc constraint the final graph cannot have more than one arc. Therefore we can rewrite $\text{NARC} = 1$ to $\text{NARC} \geq 1$ and simplify $\overline{\text{NARC}}$ to $\overline{\text{NARC}}$.

Automaton

Figure 4.180 depicts the automaton associated to the `element_matrix` constraint. Let I_k , J_k and V_k respectively be the i , the j and the v k^{th} attributes of the `MATRIX` collection. To each sextuple $(\text{INDEX_I}, \text{INDEX_J}, \text{VALUE}, I_k, J_k, V_k)$ corresponds a 0-1 signature variable S_k as well as the following signature constraint: $((\text{INDEX_I} = I_k) \wedge (\text{INDEX_J} = J_k) \wedge (\text{VALUE} = V_k)) \Leftrightarrow S_k$.

See also

`element`.

Figure 4.180: Automaton of the `element_matrix` constraintFigure 4.181: Hypergraph of the reformulation corresponding to the automaton of the `element_matrix` constraint

Key words

array constraint, data constraint, ternary constraint, matrix, automaton, automaton without counters, centered cyclic(3) constraint network(1), derived collection.

4.85 element_sparse

Origin	CHIP
Constraint	element_sparse(<i>ITEM</i> , <i>TABLE</i> , <i>DEFAULT</i>)
Usual name	element
Argument(s)	<i>ITEM</i> : collection(index – dvar, value – dvar) <i>TABLE</i> : collection(index – int, value – int) <i>DEFAULT</i> : int
Restriction(s)	required(<i>ITEM</i> , [index, value]) <i>ITEM.index</i> ≥ 1 <i>ITEM</i> = 1 required(<i>TABLE</i> , [index, value]) <i>TABLE.index</i> ≥ 1 distinct(<i>TABLE</i> , index)
Purpose	<i>ITEM.value</i> is equal to one of the entries of the table <i>TABLE</i> or to the default value <i>DEFAULT</i> if the entry <i>ITEM.index</i> does not exist in <i>TABLE</i> .
Derived Collection(s)	col(<i>DEF</i> – collection(index – int, value – int), [item(index – 0, value – <i>DEFAULT</i>)]) col $\left(\begin{array}{l} \text{TABLE_DEF} - \text{collection}(\text{index} - \text{dvar}, \text{value} - \text{dvar}), \\ \left[\begin{array}{l} \text{item}(\text{index} - \text{TABLE.index}, \text{value} - \text{TABLE.value}), \\ \text{item}(\text{index} - \text{DEF.index}, \text{value} - \text{DEF.value}) \end{array} \right] \end{array} \right)$
Arc input(s)	<i>ITEM</i> <i>TABLE_DEF</i>
Arc generator	<i>PRODUCT</i> \mapsto collection(<i>item</i> , <i>table_def</i>)
Arc arity	2
Arc constraint(s)	<ul style="list-style-type: none"> • <i>item.value</i> = <i>table_def.value</i> • <i>item.index</i> = <i>table_def.index</i> \vee <i>table_def.index</i> = 0
Graph property(ies)	$\overline{\text{NARC}} \geq 1$
Example	$\text{element_sparse} \left(\left(\begin{array}{l} \{\text{index} - 2 \text{ value} - 5\}, \\ \left\{ \begin{array}{ll} \text{index} - 1 & \text{value} - 6, \\ \text{index} - 2 & \text{value} - 5, \\ \text{index} - 4 & \text{value} - 2, \\ \text{index} - 8 & \text{value} - 9 \end{array} \right\}, 5 \end{array} \right) \right)$
	Parts (A) and (B) of Figure 4.182 respectively show the initial and final graph. Since we use the $\overline{\text{NARC}}$ graph property the final graph is outline with thick lines.
Graph model	The final graph has between one and two arc constraints: It has two arcs when the default value <i>DEFAULT</i> occurs also in the table <i>TABLE</i> ; Otherwise it has only one arc.

Automaton

Figure 4.183 depicts the automaton associated to the `element_sparse` constraint. Let `INDEX` and `VALUE` respectively be the `index` and the `value` attributes of the unique item of the `ITEM` collection. Let `INDEXi` and `VALUEi` respectively be the `index` and the `value` attributes of the i^{th} item of the `TABLE` collection. To each quintuple $(\text{INDEX}, \text{VALUE}, \text{DEFAULT}, \text{INDEX}_i, \text{VALUE}_i)$ corresponds a signature variable S_i as well as the following signature constraint:

$$\left\{ \begin{array}{l} (\text{INDEX} \neq \text{INDEX}_i \wedge \text{VALUE} \neq \text{DEFAULT}) \Leftrightarrow S_i = 0 \wedge \\ (\text{INDEX} = \text{INDEX}_i \wedge \text{VALUE} = \text{VALUE}_i) \Leftrightarrow S_i = 1 \wedge . \\ (\text{INDEX} \neq \text{INDEX}_i \wedge \text{VALUE} = \text{DEFAULT}) \Leftrightarrow S_i = 2 \end{array} \right.$$

Usage

A sometimes more compact form of the `element` constraint: We are not obliged to specify explicitly the table entries that correspond to the specified default value. This can sometimes reduce drastically memory utilisation.

Remark

The original constraint of CHIP had an additional parameter `SIZE` giving the maximum value of `ITEM.index`.

See also

`element`.

Key words

array constraint, data constraint, binary constraint, table, sparse table, sparse functional dependency, variable indexing, automaton, automaton without counters, centered cyclic(2) constraint network(1), derived collection.

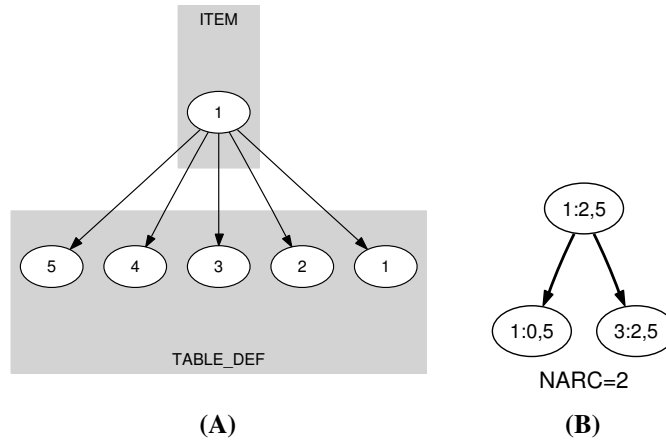


Figure 4.182: Initial and final graph of the `element_sparse` constraint

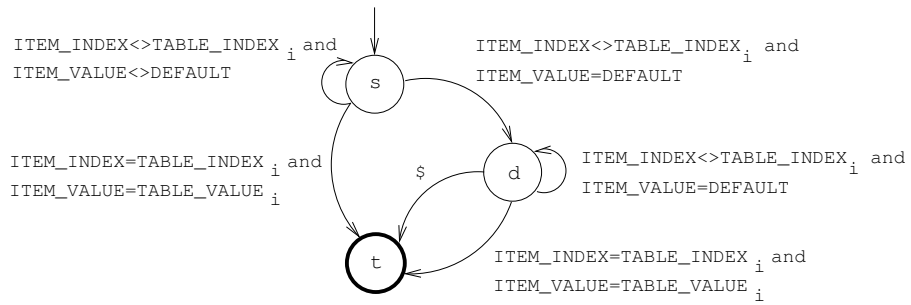


Figure 4.183: Automaton of the `element_sparse` constraint

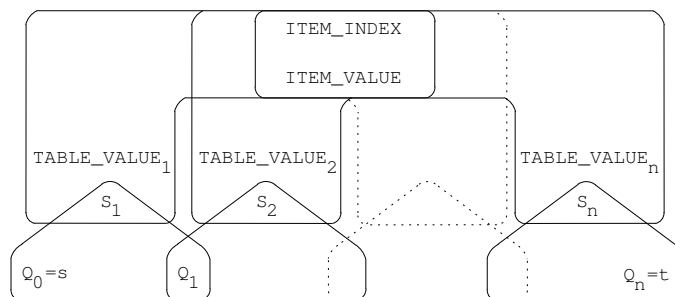


Figure 4.184: Hypergraph of the reformulation corresponding to the automaton of the `element_sparse` constraint

20030820

479

4.86 elements

Origin	Derived from element.
Constraint	elements(ITEMS, TABLE)
Argument(s)	ITEMS : collection(index - dvar, value - dvar) TABLE : collection(index - int, value - dvar)
Restriction(s)	required(ITEMS, [index, value]) ITEMS.index ≥ 1 ITEMS.index ≤ TABLE required(TABLE, [index, value]) TABLE.index ≥ 1 TABLE.index ≤ TABLE distinct(TABLE, index)
Purpose	All the items of ITEMS should be equal to one of the entries of the table TABLE.
<hr/>	
Arc input(s)	ITEMS TABLE
Arc generator	PRODUCT ↦ collection(items, table)
Arc arity	2
Arc constraint(s)	<ul style="list-style-type: none"> • items.index = table.index • items.value = table.value
Graph property(ies)	NARC = ITEMS
<hr/>	
Example	$\text{elements} \left(\begin{array}{l} \{ \text{index} - 4 \text{ value} - 9, \text{index} - 1 \text{ value} - 6 \}, \\ \left\{ \begin{array}{l} \text{index} - 1 \quad \text{value} - 6, \\ \text{index} - 2 \quad \text{value} - 9, \\ \text{index} - 3 \quad \text{value} - 2, \\ \text{index} - 4 \quad \text{value} - 9 \end{array} \right\} \end{array} \right)$
	Parts (A) and (B) of Figure 4.185 respectively show the initial and final graph. Since we use the NARC graph property, the arcs of the final graph are stressed in bold.
Signature	Since all the index attributes of TABLE collection are distinct and because of the first condition items.index = table.index of the arc constraint, a source vertex of the final graph can have at most one successor. Therefore ITEMS is the maximum number of arcs of the final graph and we can rewrite NARC = ITEMS to NARC ≥ ITEMS . So we can simplify <u>NARC</u> to <u>NARC</u> .
Usage	Used for replacing several element constraints sharing exactly the same table by one single constraint.
See also	element.
Key words	data constraint, table, shared table, functional dependency.

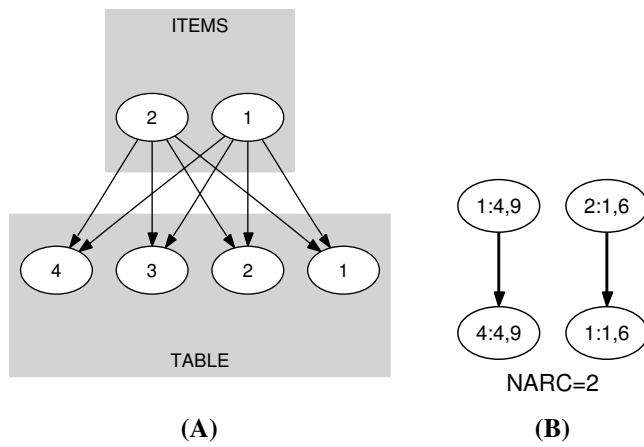


Figure 4.185: Initial and final graph of the elements constraint

4.87 elements_alldifferent

Origin	Derived from elements and alldifferent.
Constraint	elements_alldifferent(ITEMS, TABLE)
Synonym(s)	elements_alldiff, elements_alldistinct.
Argument(s)	ITEMS : collection(index – dvar, value – dvar) TABLE : collection(index – int, value – dvar)
Restriction(s)	required(ITEMS, [index, value]) ITEMS.index ≥ 1 ITEMS.index ≤ TABLE ITEMS = TABLE required(TABLE, [index, value]) TABLE.index ≥ 1 TABLE.index ≤ TABLE distinct(TABLE, index)
Purpose	All the items of the ITEMS collection should be equal to one of the entries of the table TABLE and all the variables ITEMS.index should take distinct values.
<hr/>	
Arc input(s)	ITEMS TABLE
Arc generator	<i>PRODUCT</i> ↦ collection(items, table)
Arc arity	2
Arc constraint(s)	<ul style="list-style-type: none"> • items.index = table.index • items.value = table.value
Graph property(ies)	NVERTEX = ITEMS + TABLE
<hr/>	
Example	$\text{elements_alldifferent} \left(\left(\left\{ \begin{array}{l} \text{index} - 2 \quad \text{value} - 9, \\ \text{index} - 1 \quad \text{value} - 6, \\ \text{index} - 4 \quad \text{value} - 9, \\ \text{index} - 3 \quad \text{value} - 2 \end{array} \right\}, \right. \right. \\ \left. \left. \left\{ \begin{array}{l} \text{index} - 1 \quad \text{value} - 6, \\ \text{index} - 2 \quad \text{value} - 9, \\ \text{index} - 3 \quad \text{value} - 2, \\ \text{index} - 4 \quad \text{value} - 9 \end{array} \right\} \right) \right)$
	<p>Parts (A) and (B) of Figure 4.186 respectively show the initial and final graph. Since we use the NVERTEX graph property, the vertices of the final graph are stressed in bold.</p>
Graph model	The fact that all variables ITEMS.index are pairwise different is derived from the conjunctions of the following facts:

- From the graph property $\text{NVERTEX} = |\text{ITEMS}| + |\text{TABLE}|$ it follows that all vertices of the initial graph belong also to the final graph,
- A vertex v belongs to the final graph if there is at least one constraint involving v that holds,
- From the first condition $\text{items.index} = \text{table.index}$ of the arc constraint, and from the restriction $\text{distinct}(\text{TABLE.index})$ it follows: For all vertices v generated from the collection ITEMS at most one constraint involving v holds.

Signature

Since the final graph cannot have more than $|\text{ITEMS}| + |\text{TABLE}|$ vertices one can simplify NVERTEX to NVERTEX.

Usage

Used for replacing by one single `elements_alldifferent` constraint an `alldifferent` and a set of `element` constraints having the following structure:

- The union of the index variables of the `element` constraints is equal to the set of variables of the `alldifferent` constraint.
- All the `element` constraints share exactly the same table.

For instance, the constraint given in the previous example is equivalent to the conjunction of the following set of constraints:

$$\text{alldifferent}(\{\text{var} - 2, \text{var} - 1, \text{var} - 4, \text{var} - 3\})$$

$$\text{element} \left(\left(\begin{array}{l} \{ \text{index} - 2 \quad \text{value} - 9 \}, \\ \{ \text{index} - 1 \quad \text{value} - 6, \\ \text{index} - 2 \quad \text{value} - 9, \\ \text{index} - 3 \quad \text{value} - 2, \\ \text{index} - 4 \quad \text{value} - 9 \} \end{array} \right) \right)$$

$$\text{element} \left(\left(\begin{array}{l} \{ \text{index} - 1 \quad \text{value} - 6 \}, \\ \{ \text{index} - 1 \quad \text{value} - 6, \\ \text{index} - 2 \quad \text{value} - 9, \\ \text{index} - 3 \quad \text{value} - 2, \\ \text{index} - 4 \quad \text{value} - 9 \} \end{array} \right) \right)$$

$$\text{element} \left(\left(\begin{array}{l} \{ \text{index} - 3 \quad \text{value} - 2 \}, \\ \{ \text{index} - 1 \quad \text{value} - 6, \\ \text{index} - 2 \quad \text{value} - 9, \\ \text{index} - 3 \quad \text{value} - 2, \\ \text{index} - 4 \quad \text{value} - 9 \} \end{array} \right) \right)$$

$$\text{element} \left(\left(\begin{array}{l} \{ \text{index} - 4 \quad \text{value} - 9 \}, \\ \{ \text{index} - 1 \quad \text{value} - 6, \\ \text{index} - 2 \quad \text{value} - 9, \\ \text{index} - 3 \quad \text{value} - 2, \\ \text{index} - 4 \quad \text{value} - 9 \} \end{array} \right) \right)$$

As a practical example of utilization of the `elements_alldifferent` constraint we show how to model the link between a permutation consisting of one single cycle and its expanded form. For instance, to the permutation 3, 6, 5, 2, 4, 1 corresponds the sequence

3 5 4 2 6 1. Let us note $S_1, S_2, S_3, S_4, S_5, S_6$ the permutation and $V_1 V_2 V_3 V_4 V_5 V_6$ its expanded form.

The constraint:

$$\text{elements_alldifferent} \left(\left(\left. \begin{array}{l} \text{index} - V_1 \quad \text{value} - V_2, \\ \text{index} - V_2 \quad \text{value} - V_3, \\ \text{index} - V_3 \quad \text{value} - V_4, \\ \text{index} - V_4 \quad \text{value} - V_5, \\ \text{index} - V_5 \quad \text{value} - V_6, \\ \text{index} - V_6 \quad \text{value} - V_1 \end{array} \right\} , \right. \right. \\ \left. \left. \left. \begin{array}{l} \text{index} - 1 \quad \text{value} - S_1, \\ \text{index} - 2 \quad \text{value} - S_2, \\ \text{index} - 3 \quad \text{value} - S_3, \\ \text{index} - 4 \quad \text{value} - S_4, \\ \text{index} - 5 \quad \text{value} - S_5, \\ \text{index} - 6 \quad \text{value} - S_6 \end{array} \right\} \right) \right)$$

models the fact that $S_1, S_2, S_3, S_4, S_5, S_6$ corresponds to a permutation with one single cycle. It also expresses the link between the variables $S_1, S_2, S_3, S_4, S_5, S_6$ and $V_1, V_2, V_3, V_4, V_5, V_6$.

See also

`alldifferent`, `element`.

Key words

data constraint, table, functional dependency, permutation, disequality.

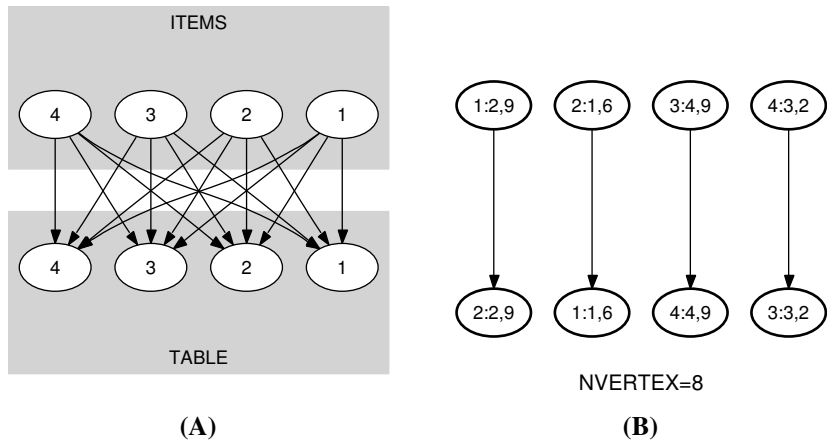


Figure 4.186: Initial and final graph of the `elements_alldifferent` constraint

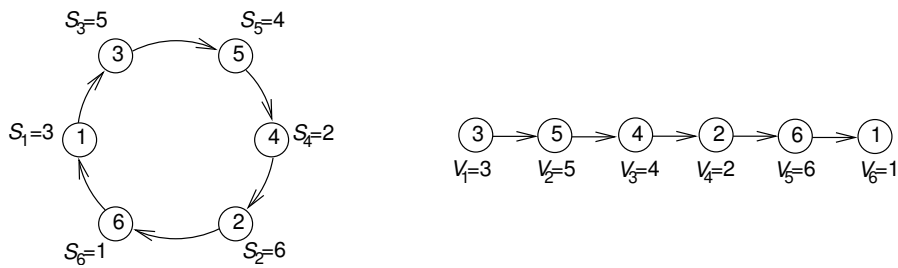


Figure 4.187: Two representations of a permutation containing one single cycle

4.88 elements_sparse

Origin	Derived from element_sparse.
Constraint	elements_sparse(ITEMS, TABLE, DEFAULT)
Argument(s)	ITEMS : collection(index - dvar, value - dvar) TABLE : collection(index - int, value - int) DEFAULT : int
Restriction(s)	required(ITEMS, [index, value]) ITEMS.index ≥ 1 required(TABLE, [index, value]) TABLE.index ≥ 1 distinct(TABLE, index)
Purpose	All the items of ITEMS should be equal to one of the entries of the table TABLE or to the default value DEFAULT if the entry ITEMS.index does not occurs among the values of the index attribute of the TABLE collection.
Derived Collection(s)	$\text{col}(\text{DEF} - \text{collection}(\text{index} - \text{int}, \text{value} - \text{int}), [\text{item}(\text{index} - 0, \text{value} - \text{DEFAULT})])$ $\text{col} \left(\begin{array}{l} \text{TABLE_DEF} - \text{collection}(\text{index} - \text{dvar}, \text{value} - \text{dvar}), \\ \left[\begin{array}{l} \text{item}(\text{index} - \text{TABLE.index}, \text{value} - \text{TABLE.index}), \\ \text{item}(\text{index} - \text{DEF.index}, \text{value} - \text{DEF.value}) \end{array} \right] \end{array} \right)$
Arc input(s)	ITEMS TABLE_DEF
Arc generator	<i>PRODUCT</i> \mapsto collection(items, table_def)
Arc arity	2
Arc constraint(s)	<ul style="list-style-type: none"> • items.value = table_def.value • items.index = table_def.index \vee table_def.index = 0
Graph property(ies)	<u>NSOURCE = ITEMS </u>

Example

$$\text{elements_sparse} \left(\begin{array}{l} \left\{ \begin{array}{l} \text{index} - 8 \quad \text{value} - 9, \\ \text{index} - 3 \quad \text{value} - 5, \\ \text{index} - 2 \quad \text{value} - 5 \end{array} \right\}, \\ \left\{ \begin{array}{l} \text{index} - 1 \quad \text{value} - 6, \\ \text{index} - 2 \quad \text{value} - 5, \\ \text{index} - 4 \quad \text{value} - 2, \\ \text{index} - 8 \quad \text{value} - 9 \end{array} \right\}, 5 \end{array} \right)$$

Parts (A) and (B) of Figure 4.188 respectively show the initial and final graph. Since we use the **NSOURCE** graph property, the vertices of the final graph are drawn with a double circle.

Graph model	An item of the ITEMS collection may have up to two successors (see for instance the third item of the ITEMS collection of the previous example). Therefore we use the graph property NSOURCE = ITEMS for enforcing the fact that each item of the ITEMS collection has at least one successor.
Signature	On the one hand note that ITEMS is equal to the number of sources of the initial graph. On the other hand observe that, in the initial graph, all the vertices which are not sources correspond to sinks. Since isolated vertices are eliminated from the final graph the sinks of the initial graph cannot become sources of the final graph. Therefore the maximum number of sources of the final graph is equal to ITEMS. We can rewrite NSOURCE = ITEMS to NSOURCE ≥ ITEMS and simplify <u>NSOURCE</u> to NSOURCE .
Usage	Used for replacing several <code>element</code> constraints sharing exactly the same sparse table by one single constraint.
See also	<code>element</code> , <code>element_sparse</code> .
Key words	data constraint, table, shared table, sparse table, sparse functional dependency, derived collection.

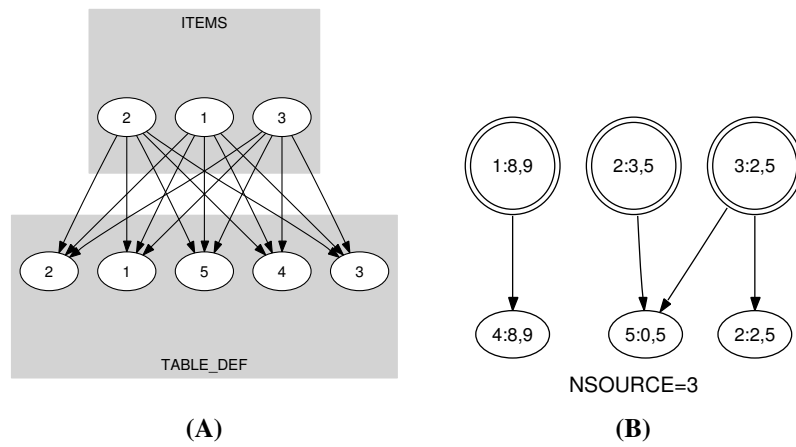


Figure 4.188: Initial and final graph of the `elements_sparse` constraint

4.89 eq_set

Origin	Used for defining <code>alldifferent_between_sets</code> .
Constraint	<code>eq_set(SET1, SET2)</code>
Argument(s)	SET1 : svar SET2 : svar
Purpose	Constraint the set SET1 to be equal to the set SET2.
Example	<code>eq_set({3, 5}, {3, 5})</code>
Used in	<code>alldifferent_between_sets</code> .
Key words	predefined constraint, binary constraint, equality, constraint involving set variables.

20030820

491

4.90 exactly

Origin	Derived from <code>atleast</code> and <code>atmost</code> .
Constraint	<code>exactly(N, VARIABLES, VALUE)</code>
Argument(s)	<code>N</code> : int <code>VARIABLES</code> : collection(var – dvar) <code>VALUE</code> : int
Restriction(s)	$N \geq 0$ $N \leq \text{VARIABLES} $ <code>required(VARIABLES, var)</code>
Purpose	Exactly N variables of the <code>VARIABLES</code> collection are assigned to value <code>VALUE</code> .
Arc input(s)	<code>VARIABLES</code>
Arc generator	<i>SELF</i> \mapsto collection(variables)
Arc arity	1
Arc constraint(s)	<code>variables.var = VALUE</code>
Graph property(ies)	<code>NARC = N</code>
Example	<code>exactly(2, {var – 4, var – 2, var – 4, var – 5}, 4)</code>

Parts (A) and (B) of Figure 4.189 respectively show the initial and final graph. Since we use the **NARC** graph property, the unary arcs of the final graph are stressed in bold. The `exactly` constraint holds since exactly 2 variables are assigned to value 4.

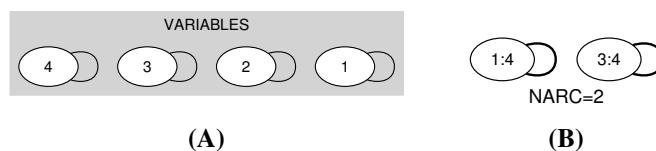


Figure 4.189: Initial and final graph of the `exactly` constraint

Graph model	Since we use a unary arc constraint (<code>VALUE</code> is fixed) we employ the <i>SELF</i> arc generator in order to produce a graph with a single loop on each vertex.
Automaton	Figure 4.190 depicts the automaton associated to the <code>exactly</code> constraint. To each variable VAR_i of the collection <code>VARIABLES</code> corresponds a 0-1 signature variable S_i . The following signature constraint links VAR_i and S_i : $\text{VAR}_i = \text{VALUE} \Leftrightarrow S_i$.

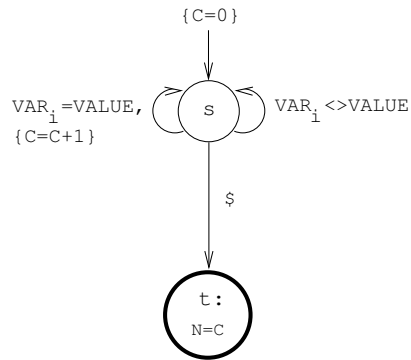


Figure 4.190: Automaton of the exactly constraint

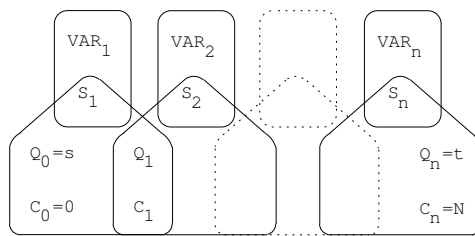


Figure 4.191: Hypergraph of the reformulation corresponding to the automaton of the exactly constraint

See also atleast, atmost, among.

Key words value constraint, counting constraint, automaton, automaton with counters,
alpha-acyclic constraint network(2).

20040807

495

4.91 global_cardinality

Origin	CHARME
Constraint	global_cardinality(VARIABLES, VALUES)
Synonym(s)	distribute, distribution, gcc, card_var_gcc, egcc.
Argument(s)	VARIABLES : collection(var – dvar) VALUES : collection(val – int, noccurrence – dvar)
Restriction(s)	required(VARIABLES, var) required(VALUES, [val, noccurrence]) distinct(VALUES, val) VALUES.noccurrence \geq 0 VALUES.noccurrence \leq VARIABLES
Purpose	Each value VALUES[i].val ($1 \leq i \leq VALUES $) should be taken by exactly VALUES[i].noccurrence variables of the VARIABLES collection.
	For all items of VALUES:
Arc input(s)	VARIABLES
Arc generator	<i>SELF</i> \mapsto collection(variables)
Arc arity	1
Arc constraint(s)	variables.var = VALUES.val
Graph property(ies)	NVERTEX = VALUES.noccurrence

Example global_cardinality $\left(\left(\left\{ \begin{array}{l} \text{var} - 3, \\ \text{var} - 3, \\ \text{var} - 8, \\ \text{var} - 6 \end{array} \right\}, \right. \right. \left. \left. \left\{ \begin{array}{ll} \text{val} - 3 & \text{noccurrence} - 2, \\ \text{val} - 5 & \text{noccurrence} - 0, \\ \text{val} - 6 & \text{noccurrence} - 1 \end{array} \right\} \right) \right)$

The constraint holds since values 3, 5 and 6 are respectively used 2, 0 and 1 times and since no constraint was specified for value 8. Part (A) of Figure 4.192 shows the initial graphs associated to each value 3, 5 and 6 of the VALUES collection. Part (B) of Figure 4.192 shows the two final graphs respectively associated to values 3 and 6 which are both assigned to the variables of the VARIABLES collection (since value 5 is not assigned to any variable of the VARIABLES collection the final graph associated to value 5 is empty). Since we use the **NVERTEX** graph property, the vertices of the final graphs are stressed in bold.

Graph model Since we want to express one unary constraint for each value we use the “For all items of VALUES” iterator.

Automaton Figure 4.193 depicts the automaton associated to the `global_cardinality` constraint. To each item of the collection `VARIABLES` corresponds a signature variable S_i , which is equal to 0. To each item of the collection `VALUES` corresponds a signature variable $S_{i+|VARIABLES|}$, which is equal to 1.

Usage We show how to use the `global_cardinality` constraint in order to model the *magic series* problem [113, page 155] with one single `global_cardinality` constraint. A non-empty finite series $S = (s_0, s_1, \dots, s_n)$ is *magic* if and only if there are s_i occurrences of i in S for each integer i ranging from 0 to n . This leads to the following constraint:

$$\text{global_cardinality} \left(\left\{ \begin{array}{l} \{ \text{var} - s_0, \text{var} - s_1, \dots, \text{var} - s_n \}, \\ \left\{ \begin{array}{ll} \text{val} - 0 & \text{noccurrence} - s_0, \\ \text{val} - 1 & \text{noccurrence} - s_1, \\ & \vdots \\ \text{val} - n & \text{noccurrence} - s_n \end{array} \right\} \end{array} \right\} \right)$$

Remark This is a generalized form of the original `global_cardinality` constraint: In the original `global_cardinality` constraint [19], one specifies for each value its minimum and maximum number of occurrences; Here we give for each value v a domain variable which indicates how many time value v is effectively used. By setting the minimum and maximum values of this variable to the appropriate constants we can express the same thing as in the original `global_cardinality` constraint. However, as shown in the *magic series* problem, we can also use this variable in other constraints.

A last difference with the original `global_cardinality` constraint comes from the fact that there is no constraint on the values which are not mentioned in the `VALUES` collection. In the original `global_cardinality` these values could not be assigned to the variables of the `VARIABLES` collection.

Within [34] the `global_cardinality` constraint is called *distribution*. Within [80] the `global_cardinality` constraint is called *card_var_gcc*. Within [114] the `global_cardinality` constraint is called *egcc* or *rgcc*. This later case corresponds to the fact that some variables are duplicated within the `VARIABLES` collection.

W.-J. van Hoeve et al. present two soft versions of the `global_cardinality` constraint in [12].

Algorithm A flow algorithm that handles the original `global_cardinality` constraint is described in [19]. The two approaches that were used to design bound-consistency algorithms for



Figure 4.192: Initial and final graph of the `global_cardinality` constraint

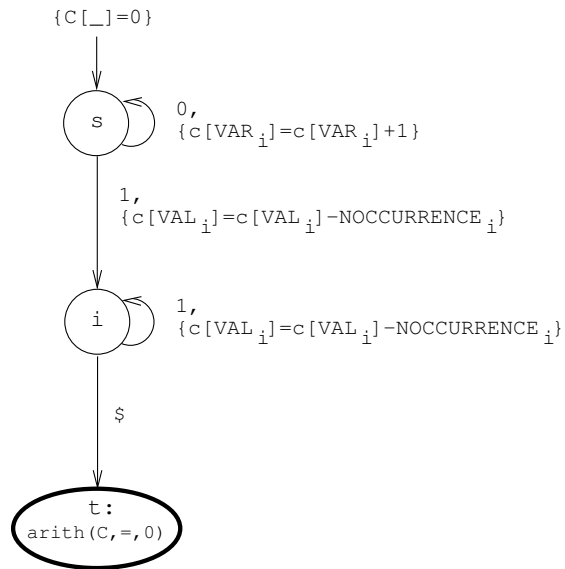
`alldifferent` were generalized for the `global_cardinality` constraint. The algorithm in [115] identifies Hall intervals and the one in [24] exploits convexity to achieve a fast implementation of the flow-based arc-consistency algorithm. The later algorithm can also compute bound-consistency for the count variables [116]. An improved algorithm for achieving arc-consistency is described in [27]. In the same paper, it is shown that it is NP-hard to compute arc-consistency for the count variables.

See also

`among`, `count`, `nvalue`, `max_nvalue`, `min_nvalue`, `global_cardinality_with_costs`, `symmetric_gcc`, `symmetric_cardinality`, `colored_matrix`, `same_and_global_cardinality`.

Key words

value constraint, assignment, magic series, Hall interval, bound-consistency, flow, duplicated variables, automaton, automaton with array of counters.

Figure 4.193: Automaton of the `global_cardinality` constraint

4.92 global_cardinality_low_up

Origin	Used for defining <code>sliding_distribution</code> .
Constraint	<code>global_cardinality_low_up(VARIABLES, VALUES)</code>
Argument(s)	<p><code>VARIABLES</code> : <code>collection(var - dvar)</code></p> <p><code>VALUES</code> : <code>collection(val - int, omin - int, omax - int)</code></p>
Restriction(s)	<p><code>required(VARIABLES, var)</code></p> <p><code> VALUES > 0</code></p> <p><code>required(VALUES, [val, omin, omax])</code></p> <p><code>distinct(VALUES, val)</code></p> <p><code>VALUES.omin ≥ 0</code></p> <p><code>VALUES.omax ≤ VARIABLES </code></p> <p><code>VALUES.omin ≤ VALUES.omax</code></p>
Purpose	<p>Each value <code>VALUES[i].val</code> ($1 \leq i \leq VALUES$) should be taken by at least <code>VALUES[i].omin</code> and at most <code>VALUES[i].omax</code> variables of the <code>VARIABLES</code> collection.</p>
	For all items of <code>VALUES</code> :
Arc input(s)	<code>VARIABLES</code>
Arc generator	<code>SELF</code> \mapsto <code>collection(variables)</code>
Arc arity	1
Arc constraint(s)	<code>variables.var = VALUES.val</code>
Graph property(ies)	<ul style="list-style-type: none"> • <code>NVERTEX ≥ VALUES.omin</code> • <code>NVERTEX ≤ VALUES.omax</code>

Example	$\text{global_cardinality_low_up} \left(\left(\left\{ \begin{array}{l} \text{var} - 3, \\ \text{var} - 3, \\ \text{var} - 8, \\ \text{var} - 6 \end{array} \right\}, \left\{ \begin{array}{lll} \text{val} - 3 & \text{omin} - 2 & \text{omax} - 3, \\ \text{val} - 5 & \text{omin} - 0 & \text{omax} - 1, \\ \text{val} - 6 & \text{omin} - 1 & \text{omax} - 2 \end{array} \right\} \right) \right)$
----------------	---

The constraint holds since values 3, 5 and 6 are respectively used 2, 0 and 1 times and since no constraint was specified for value 8. Part (A) of Figure 4.192 shows the initial graphs associated to each value 3, 5 and 6 of the `VALUES` collection. Part (B) of Figure 4.192 shows the two final graphs respectively associated to values 3 and 6 which are both assigned to the variables of the `VARIABLES` collection (since value 5 is not assigned to any variable of the `VARIABLES` collection the final graph associated to value 5 is empty). Since we use the `NVERTEX` graph property, the vertices of the final graphs are stressed in bold.

Graph model	Since we want to express one unary constraint for each value we use the “For all items of VALUES” iterator.
Algorithm	[19].
Used in	<code>sliding_distribution</code> .
See also	<code>global_cardinality</code> , <code>sliding_distribution</code> .
Key words	value constraint, assignment, flow.

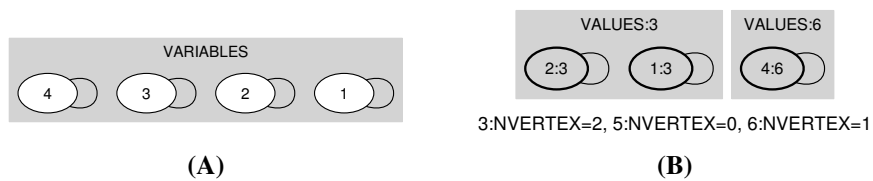


Figure 4.194: Initial and final graph of the `global_cardinality_low_up` constraint

4.93 global_cardinality_with_costs

Origin	[117]
Constraint	global_cardinality_with_costs(VARIABLES, VALUES, MATRIX, COST)
Synonym(s)	gccc, cost_gcc.
Argument(s)	VARIABLES : collection(var – dvar) VALUES : collection(val – int, noccurrence – dvar) MATRIX : collection(i – int, j – int, c – int) COST : dvar

Restriction(s)	required(VARIABLES, var) required(VALUES, [val, noccurrence]) distinct(VALUES, val) VALUES.noccurrence ≥ 0 VALUES.noccurrence \leq VARIABLES required(MATRIX, [i, j, c]) increasing_seq(MATRIX, [i, j]) MATRIX.i ≥ 1 MATRIX.i \leq VARIABLES MATRIX.j ≥ 1 MATRIX.j \leq VALUES MATRIX = VARIABLES * VALUES
-----------------------	---

Purpose

Each value VALUES[i].val should be taken by exactly VALUES[i].noccurrence variables of the VARIABLES collection. In addition the COST of an assignment is equal to the sum of the elementary costs associated to the fact that we assign the i^{th} variable of the VARIABLES collection to the j^{th} value of the VALUES collection. These elementary costs are given by the MATRIX collection.

For all items of VALUES:

Arc input(s)	VARIABLES
Arc generator	<i>SELF</i> \mapsto collection(variables)
Arc arity	1
Arc constraint(s)	variables.var = VALUES.val
Graph property(ies)	<u>NVERTEX = VALUES.noccurrence</u>
Arc input(s)	VARIABLES VALUES
Arc generator	<i>PRODUCT</i> \mapsto collection(variables, values)

Arc arity 2
Arc constraint(s) variables.var = values.val
Graph property(ies) SUM_WEIGHT_ARC(MATRIX[(variables.key - 1) * |VALUES| + values.key].c) = COST

Example global_cardinality_with_costs

$$\left(\left(\left\{ \begin{array}{l} \text{var} - 3, \\ \text{var} - 3, \\ \text{var} - 3, \\ \text{var} - 6 \end{array} \right\}, \left\{ \begin{array}{ll} \text{val} - 3 & \text{noccurrence} - 3, \\ \text{val} - 5 & \text{noccurrence} - 0, \\ \text{val} - 6 & \text{noccurrence} - 1 \end{array} \right\}, \left\{ \begin{array}{lll} i - 1 & j - 1 & c - 4, \\ i - 1 & j - 2 & c - 1, \\ i - 1 & j - 3 & c - 7, \\ i - 2 & j - 1 & c - 1, \\ i - 2 & j - 2 & c - 0, \\ i - 2 & j - 3 & c - 8, \\ i - 3 & j - 1 & c - 3, \\ i - 3 & j - 2 & c - 2, \\ i - 3 & j - 3 & c - 1, \\ i - 4 & j - 1 & c - 0, \\ i - 4 & j - 2 & c - 0, \\ i - 4 & j - 3 & c - 6 \end{array} \right\}, 14 \right) \right)$$

Parts (A) and (B) of Figure 4.195 respectively show the initial and final graph associated to the second graph constraint.

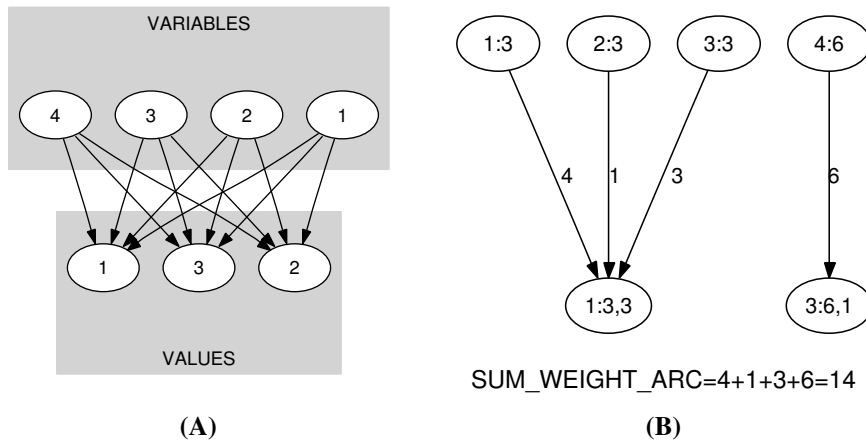


Figure 4.195: Initial and final graph of the global_cardinality_with_costs constraint

Graph model The first graph constraint enforces each value of the VALUES collection to be taken by a specific number of variables of the VARIABLES collection. It is identical to the graph

constraint used in the `global_cardinality` constraint. The second graph constraint expresses the fact that the `COST` variable is equal to the sum of the elementary costs associated to each variable-value assignment. All these elementary costs are recorded in the `MATRIX` collection. More precisely, the cost c_{ij} is recorded in the attribute `c` of the $((i - 1) \cdot |\text{VALUES}| + j)^{\text{th}}$ entry of the `MATRIX` collection. This is ensured by the `increasing` restriction which enforces the fact that the items of the `MATRIX` collection are sorted in lexicographically increasing order according to attributes `i` and `j`.

Usage

A classical utilisation of the `global_cardinality_with_costs` constraint corresponds to the following assignment problem. We have a set of persons \mathcal{P} as well as a set of jobs \mathcal{J} to perform. Each job requires a number of persons restricted to a specified interval. In addition each person p has to be assigned to one specific job taken from a subset \mathcal{J}_p of \mathcal{J} . There is a cost C_{pj} associated to the fact that person p is assigned to job j . The previous problem is modelled with one single `global_cardinality_with_costs` constraint where the persons and the jobs respectively correspond to the items of the `VARIABLES` and `VALUES` collection.

The `global_cardinality_with_costs` constraint can also be used for modelling a conjunction `alldifferent`(X_1, X_2, \dots, X_n) and $\alpha_1 \cdot X_1 + \alpha_2 \cdot X_2 + \dots + \alpha_n \cdot X_n = \text{COST}$. For this purpose we set the domain of the `noccurrence` variables to $\{0, 1\}$ and the cost attribute `c` of a variable X_i and one of its potential value j to $\alpha_i \cdot j$. In practice this can be used for the *magic squares* and the *magic hexagon* problems where all the α_i are set to 1.

Algorithm

[20]

See also`global_cardinality`, `weighted_partial_alldiff`.**Key words**

cost filtering constraint, assignment, cost matrix, weighted assignment, scalar product, magic square, magic hexagon.

20030820

505

4.94 global_contiguity

Origin	[35]
Constraint	global_contiguity(VARIABLES)
Argument(s)	VARIABLES : collection(var – dvar)
Restriction(s)	required(VARIABLES, var) VARIABLES.var ≥ 0 VARIABLES.var ≤ 1
Purpose	Enforce all variables of the VARIABLES collection to be assigned to 0 or 1. In addition, all variables assigned to value 1 appear contiguously.
<hr/>	
Arc input(s)	VARIABLES
Arc generator	PATH \mapsto collection(variables1, variables2) LOOP \mapsto collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	<ul style="list-style-type: none"> • variables1.var = variables2.var • variables1.var = 1
Graph property(ies)	NCC ≤ 1
<hr/>	
Example	$\text{global_contiguity} \left(\left(\left\{ \begin{array}{l} \text{var} - 0, \\ \text{var} - 1, \\ \text{var} - 1, \\ \text{var} - 0 \end{array} \right\} \right) \right)$ <p>Parts (A) and (B) of Figure 4.196 respectively show the initial and final graph. The global_contiguity constraint holds since the final graph does not contain more than one connected component. This connected component corresponds to 2 contiguous variables which are both assigned to 1.</p>
Graph model	Each connected component of the final graph corresponds to one set of contiguous variables that all take value 1.
Automaton	Figure 4.197 depicts the automaton associated to the global_contiguity constraint. To each variable VAR _i of the collection VARIABLES corresponds a signature variable, which is equal to VAR _i . There is no signature constraint.
Usage	The paper [35] introducing this constraint refers to hardware configuration problems.
Algorithm	A filtering algorithm for this constraint is described in [35].
See also	group, inflexion.
Key words	connected component, convex, Berge-acyclic constraint network, automaton, automaton without counters.

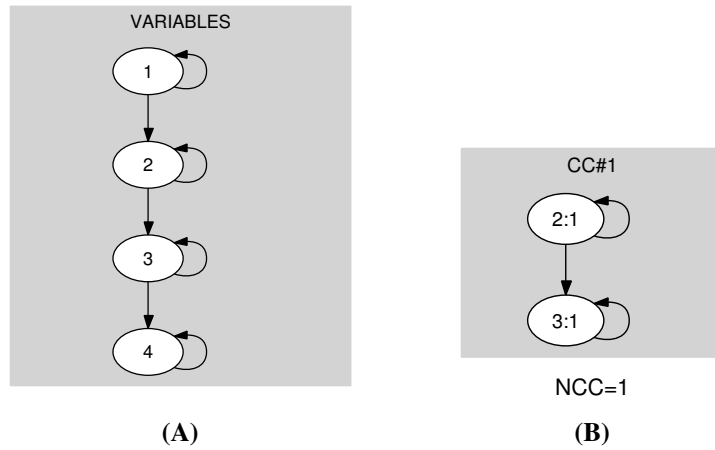


Figure 4.196: Initial and final graph of the global_contiguity constraint

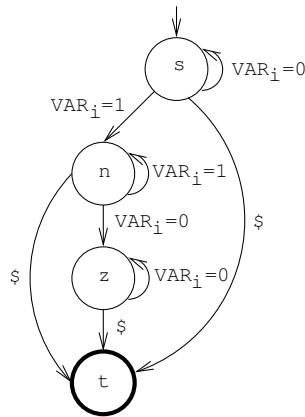


Figure 4.197: Automaton of the global_contiguity constraint

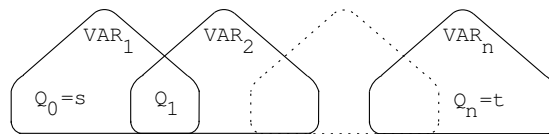


Figure 4.198: Hypergraph of the reformulation corresponding to the automaton of the global_contiguity constraint

4.95 golomb

Origin	Inspired by [118].
Constraint	golomb(VARIABLES)
Argument(s)	VARIABLES : collection(var - dvar)
Restriction(s)	required(VARIABLES, var) VARIABLES.var ≥ 0
Purpose	Enforce all differences $X_i - X_j$ between two variables X_i and X_j ($i > j$) of the collection VARIABLES to be distinct.
Derived Collection(s)	col $\left(\begin{array}{l} \text{PAIRS} - \text{collection}(x - \text{dvar}, y - \text{dvar}), \\ [\text{>} - \text{item}(x - \text{VARIABLES.var}, y - \text{VARIABLES.var})] \end{array} \right)$
Arc input(s)	PAIRS
Arc generator	CLIQUE \mapsto collection(pairs1, pairs2)
Arc arity	2
Arc constraint(s)	pairs1.y - pairs1.x = pairs2.y - pairs2.x
Graph property(ies)	MAX_NSCC ≤ 1
Example	golomb({var - 0, var - 1, var - 4, var - 6})
	<p>Parts (A) and (B) of Figure 4.199 respectively show the initial and final graph. Since we use the MAX_NSCC graph property we show one of the largest strongly connected component of the final graph. The constraint holds since all the strongly connected components have at most one vertex: the differences 1, 2, 3, 4, 5, 6 that one can construct from the values 0, 1, 4, 6 assigned to the variables of the VARIABLES collection are all distinct. Figure 4.200 gives a graphical interpretation of the solution given in the example in term of a graph: Each vertex corresponds to a variable, while each arc depicts a difference between two variables. One can observe that these differences are all distinct.</p>
Graph model	When applied on the collection of items {VAR1, VAR2, VAR3, VAR4}, the generator of derived collection generates the following collection of items: {VAR2 VAR1, VAR3 VAR1, VAR3 VAR2, VAR4 VAR1, VAR4 VAR2, VAR4 VAR3}. Note that we use a binary arc constraint between two vertices and that this binary constraint involves four variables.
Usage	This constraint refers to the Golomb ruler problem. We quote the definition from [119]: "A Golomb ruler is a set of integers (marks) $a_1 < \dots < a_k$ such that all the differences $a_i - a_j$ ($i > j$) are distinct".
Remark	Different constraints models for the Golomb ruler problem were presented in [120].

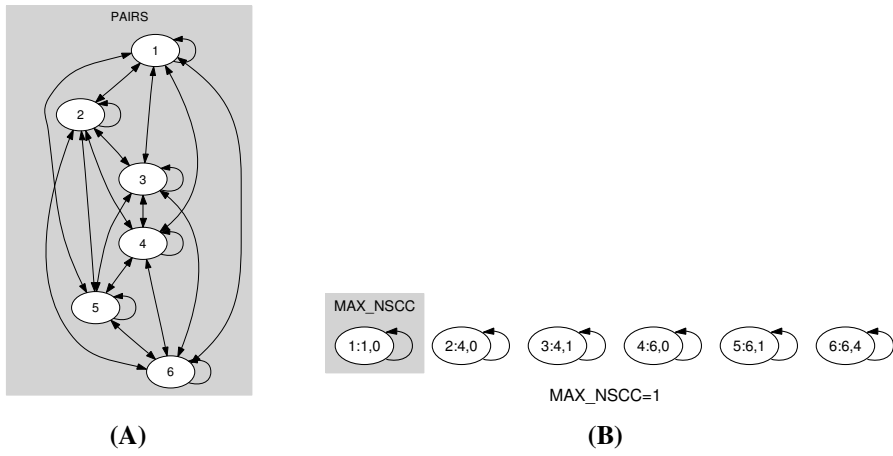


Figure 4.199: Initial and final graph of the go1omb constraint

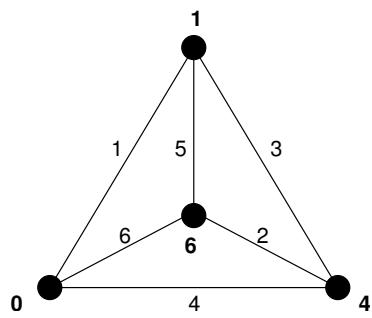


Figure 4.200: Graphical representation of the solution 0,1,4,6

- Algorithm** At a first glance, one could think that, because it looks so similar to the `alldifferent` constraint, we could have a perfect polynomial filtering algorithm. However this is not true since one retrieves the *same* variable in different vertices of the graph. This leads to the fact that one has incompatible arcs in the bipartite graph (the two classes of vertices correspond to the pair of variables and to the fact that the difference between two pairs of variables takes a specific value). However one can still reuse a similar filtering algorithm as for the `alldifferent` constraint, but this will not lead to perfect pruning.
- See also** `alldifferent`.
- Key words** Golomb ruler, disequality, difference, derived collection.

4.96 graph_crossing

Origin	N. Beldiceanu
Constraint	graph_crossing(NCROSS, NODES)
Argument(s)	NCROSS : dvar NODES : collection(succ - dvar, x - int, y - int)
Restriction(s)	NCROSS ≥ 0 required(NODES, [succ, x, y]) NODES.succ ≥ 1 NODES.succ ≤ NODES
Purpose	NCROSS is the number of proper intersections between line-segments, where each line-segment is an arc of the directed graph defined by the arc linking a node and its unique successor.
Arc input(s)	NODES
Arc generator	<i>CLIQUE</i> (<) ↦ collection(n1, n2)
Arc arity	2
Arc constraint(s)	<ul style="list-style-type: none"> • $\max(n1.x, NODES[n1.succ].x) \geq \min(n2.x, NODES[n2.succ].x)$ • $\max(n2.x, NODES[n2.succ].x) \geq \min(n1.x, NODES[n1.succ].x)$ • $\max(n1.y, NODES[n1.succ].y) \geq \min(n2.y, NODES[n2.succ].y)$ • $\max(n2.y, NODES[n2.succ].y) \geq \min(n1.y, NODES[n1.succ].y)$ • $(n2.x - NODES[n1.succ].x) * (NODES[n1.succ].y - n1.y) - (NODES[n1.succ].x - n1.x) * (n2.y - NODES[n1.succ].y) \neq 0$ • $(NODES[n2.succ].x - NODES[n1.succ].x) * (n2.y - n1.y) - (n2.x - n1.x) * (NODES[n2.succ].y - NODES[n1.succ].y) \neq 0$ • $\left. \begin{array}{l} \text{sign} \left(\begin{array}{l} (n2.x - NODES[n1.succ].x) * (NODES[n1.succ].y - n1.y) - \\ (NODES[n1.succ].x - n1.x) * (n2.y - NODES[n1.succ].y) \end{array} \right) \\ \text{sign} \left(\begin{array}{l} (NODES[n2.succ].x - NODES[n1.succ].x) * (n2.y - n1.y) - \\ (n2.x - n1.x) * (NODES[n2.succ].y - NODES[n1.succ].y) \end{array} \right) \end{array} \right) \neq$
Graph property(ies)	NARC = NCROSS

Example	graph_crossing	2,	$\left(\begin{array}{c} \left(\begin{array}{l} \text{succ} - 1 \quad x - 4 \quad y - 7, \\ \text{succ} - 1 \quad x - 2 \quad y - 5, \\ \text{succ} - 1 \quad x - 7 \quad y - 6, \\ \text{succ} - 2 \quad x - 1 \quad y - 2, \\ \text{succ} - 3 \quad x - 2 \quad y - 2, \\ \text{succ} - 2 \quad x - 5 \quad y - 3, \\ \text{succ} - 3 \quad x - 8 \quad y - 2, \\ \text{succ} - 9 \quad x - 6 \quad y - 2, \\ \text{succ} - 10 \quad x - 10 \quad y - 6, \\ \text{succ} - 8 \quad x - 10 \quad y - 1 \end{array} \right) \end{array} \right)$
----------------	----------------	----	--

Parts (A) and (B) of Figure 4.201 respectively show the initial and final graph. Since we use the **NARC** graph property, the arcs of the final graph are stressed in bold. Each arc of the final graph corresponds to a proper intersection between two line-segments. Figure 4.202 shows the line-segments associated to the **NODES** collection. One can observe the following line-segments intersection:

- Arcs $8 \rightarrow 9$ and $7 \rightarrow 3$ cross,
- Arcs $5 \rightarrow 3$ and $7 \rightarrow 3$ cross also.

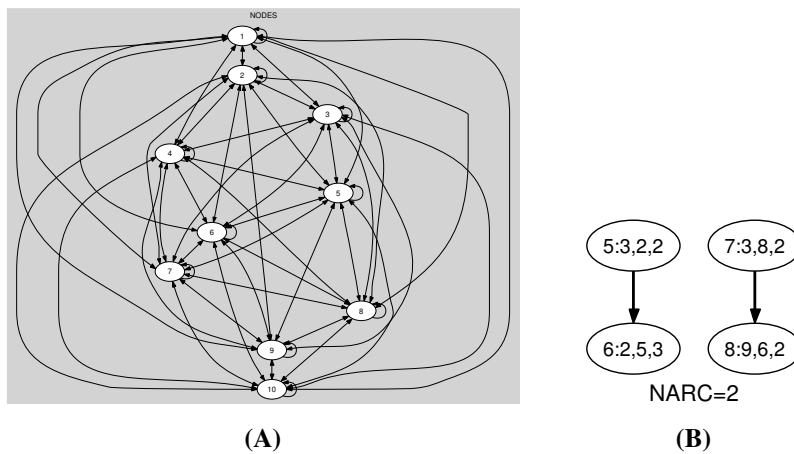


Figure 4.201: Initial and final graph of the `graph_crossing` constraint

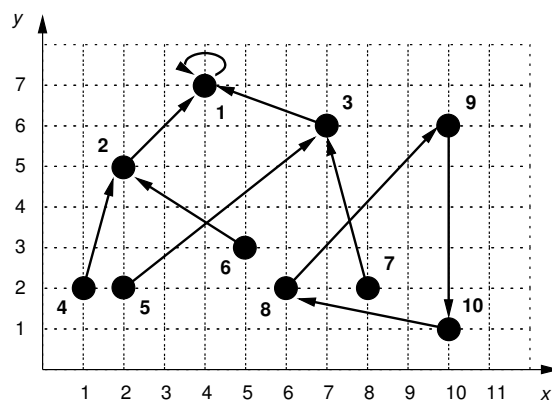


Figure 4.202: A graph covering with 2 line-segments intersections

Graph model

Each node is described by its coordinates x and y , and by its successor `succ` in the final covering. Note that the coordinates are initially fixed. We use the arc generator `CLIQUE(<)` in order to avoid counting twice the same line-segment crossing.

- Usage** This is a general crossing constraint that can be used in conjunction with one graph covering constraint such as *cycle*, *tree* or *map*. In many practical problems ones want not only to cover a graph with specific patterns but also to avoid too much crossing between the arcs of the final graph.
- Remark** We did not give a specific crossing constraint for each graph covering constraint. We feel that it is better to start first with a more general constraint before going in the specificity of the pattern that is used for covering the graph.
- See also** *crossing*, *two_layer_edge_crossing*, *cycle*, *tree*, *map*.
- Key words** geometrical constraint, line-segments intersection.

20000128

515

4.97 group

Origin	CHIP
Constraint	group(NGROUP, MIN_SIZE, MAX_SIZE, MIN_DIST, MAX_DIST, NVAL, VARIABLES, VALUES)
Argument(s)	<pre> NGROUP : dvar MIN_SIZE : dvar MAX_SIZE : dvar MIN_DIST : dvar MAX_DIST : dvar NVAL : dvar VARIABLES : collection(var - dvar) VALUES : collection(val - int) </pre>

Restriction(s)	<pre> NGROUP ≥ 0 MIN_SIZE ≥ 0 MAX_SIZE ≥ MIN_SIZE MIN_DIST ≥ 0 MAX_DIST ≥ MIN_DIST NVAL ≥ 0 required(VARIABLES, var) required(VALUES, val) distinct(VALUES, val) </pre>
-----------------------	---

Purpose

Let n be the number of variables of the collection VARIABLES. Let X_i, X_{i+1}, \dots, X_j ($1 \leq i \leq j \leq n$) be consecutive variables of the collection of variables VARIABLES such that all the following conditions simultaneously apply:

- All variables X_i, \dots, X_j take their value in the set of values VALUES,
- $i = 1$ or X_{i-1} does not take a value in VALUES,
- $j = n$ or X_{j+1} does not take a value in VALUES.

We call such a set of variables a *group*. The constraint group is true if all the following conditions hold:

- There are exactly NGROUP groups of variables,
- MIN_SIZE is the number of variables of the smallest group,
- MAX_SIZE is the number of variables of the largest group,
- MIN_DIST is the minimum number of variables between two consecutive groups or between one border and one group,
- MAX_DIST is the maximum number of variables between two consecutive groups or between one border and one group,
- NVAL is the number of variables that take their value in the set of values VALUES.

Arc input(s)	VARIABLES
Arc generator	<pre> PATH ↦ collection(variables1, variables2) LOOP ↦ collection(variables1, variables2) </pre>

Arc arity	2
Arc constraint(s)	<ul style="list-style-type: none"> • <code>in(variables1.var, VALUES)</code> • <code>in(variables2.var, VALUES)</code>
Graph property(ies)	<ul style="list-style-type: none"> • <code>NCC = NGROUP</code> • <code>MIN_NCC = MIN_SIZE</code> • <code>MAX_NCC = MAX_SIZE</code> • <code>NVERTEX = NVAL</code>
<hr/>	
Arc input(s)	VARIABLES
Arc generator	$PATH \mapsto \text{collection}(\text{variables1}, \text{variables2})$ $LOOP \mapsto \text{collection}(\text{variables1}, \text{variables2})$
Arc arity	2
Arc constraint(s)	<ul style="list-style-type: none"> • <code>not_in(variables1.var, VALUES)</code> • <code>not_in(variables2.var, VALUES)</code>
Graph property(ies)	<ul style="list-style-type: none"> • <code>MIN_NCC = MIN_DIST</code> • <code>MAX_NCC = MAX_DIST</code>
<hr/>	

Example

$$\text{group} \left(\left(\left(2, 1, 2, 2, 4, 3, \right. \right. \left. \left. \left. \left. \left. \begin{array}{l} \text{var} - 2, \\ \text{var} - 8, \\ \text{var} - 1, \\ \text{var} - 7, \\ \text{var} - 4, \\ \text{var} - 5, \\ \text{var} - 1, \\ \text{var} - 1, \\ \text{var} - 1 \end{array} \right. \right. \right. \right. \right. \right. \left. \left. \left. \left. \left. \begin{array}{l} \text{val} - 0, \\ \text{val} - 2, \\ \text{val} - 4, \\ \text{val} - 6, \\ \text{val} - 8 \end{array} \right. \right. \right. \right. \right. \right)$$

The previous constraint holds since:

- The final graph of the first graph constraint has two connected components. Therefore the number of groups `NGROUP` is equal to two.
- The number of vertices of the smallest connected component of the final graph of the first graph constraint is equal to one. Therefore `MIN_SIZE` is equal to one.
- The number of vertices of the largest connected component of the final graph of the first graph constraint is equal to two. Therefore `MAX_SIZE` is equal to two.
- The number of vertices of the smallest connected component of the final graph of the second graph constraint is equal to two. Therefore `MIN_DIST` is equal to two.
- The number of vertices of the largest connected component of the final graph of the second graph constraint is equal to four. Therefore `MAX_DIST` is equal to four.

- The number of vertices of the final graph of the first graph constraint is equal to three. Therefore *NVAL* is equal to three.

Parts (A) and (B) of Figure 4.203 respectively show the initial and final graph associated to the first graph constraint. Since we use the **NVERTEX** graph property, the vertices of the final graph are stressed in bold. In addition, since we use the **MIN_NCC** and the **MAX_NCC** graph properties, we also show the smallest and largest connected components of the final graph.

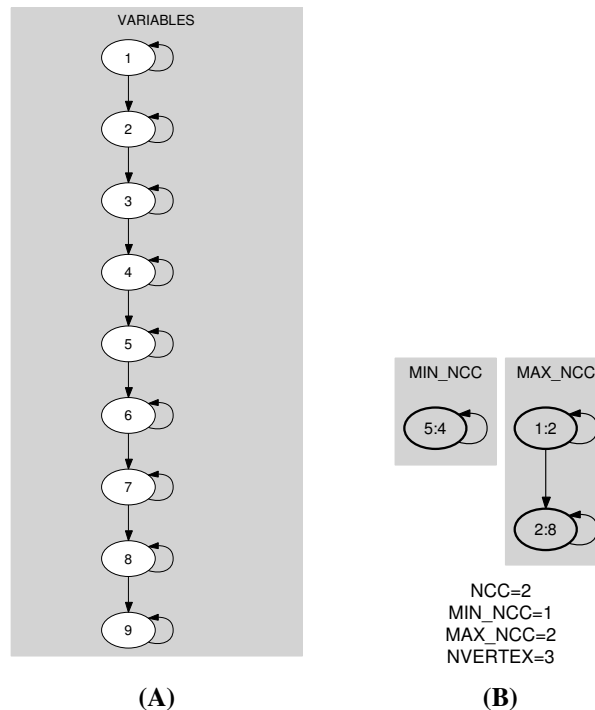


Figure 4.203: Initial and final graph of the group constraint

Graph model

We use two graph constraints for modelling the group constraint: A first one for specifying the constraints on *NGROUP*, *MIN_SIZE*, *MAX_SIZE* and *NVAL*, and a second one for stating the constraints on *MIN_DIST* and *MAX_DIST*. In order to generate the initial graph related to the first graph constraint we use:

- The arc generators *PATH* and *LOOP*,
- The binary constraint $\text{variables1.var} \in \text{VALUES} \wedge \text{variables2.var} \in \text{VALUES}$.

This produces an initial graph depicted in part (A) of Figure 4.203. We use *PATH LOOP* and the binary constraint $\text{variables1.var} \in \text{VALUES} \wedge \text{variables2.var} \in \text{VALUES}$ in order to catch the two following situations:

- A binary constraint has to be used in order to get the notion of group: *Consecutive* variables that take their value in *VALUES*.

- If we only use *PATH* then we would lose the groups that are composed from one single variable since the predecessor and the successor arc would be destroyed; this is why we use also the *LOOP* arc generator.

Automaton

Figures 4.204, 4.206, 4.207, 4.209, 4.210 and 4.212 depict the different automata associated to the group constraint. For the automata that respectively compute *NGROUP*, *MIN_SIZE*, *MAX_SIZE*, *MIN_DIST*, *MAX_DIST* and *NVAL* we have a 0-1 signature variable S_i for each variable VAR_i of the collection *VARIABLES*. The following signature constraint links VAR_i and S_i : $VAR_i \in VALUES \Leftrightarrow S_i$.

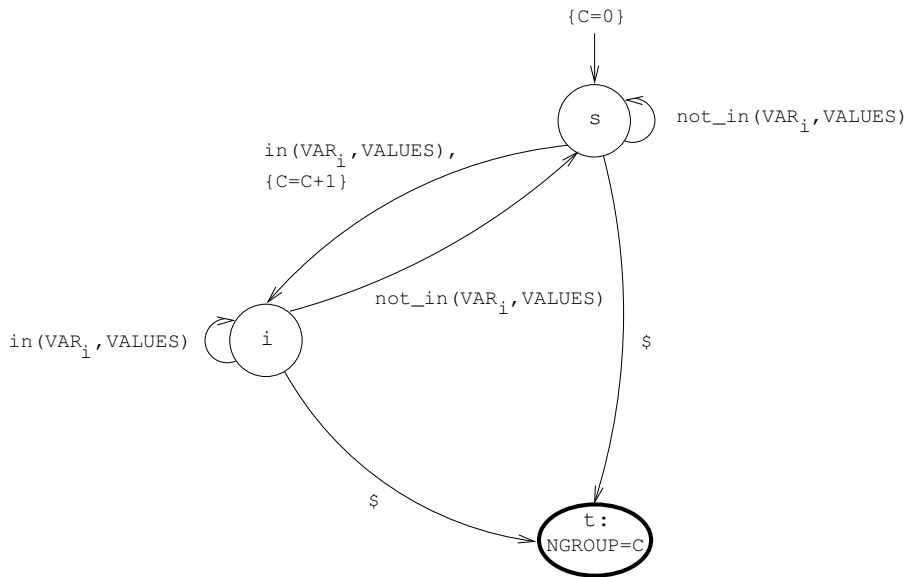


Figure 4.204: Automaton for the *NGROUP* parameter of the group constraint

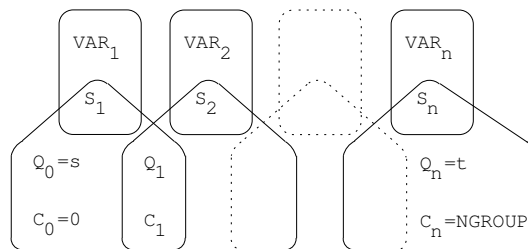


Figure 4.205: Hypergraph of the reformulation corresponding to the automaton of the *NGROUP* parameter of the group constraint

Usage

A typical use of the group constraint in the context of timetabling is as follow: The value of the i^{th} variable of the *VARIABLES* collection corresponds to the type of shift (i.e. night, morning, afternoon, rest) performed by a specific persons on day i . A complete period of

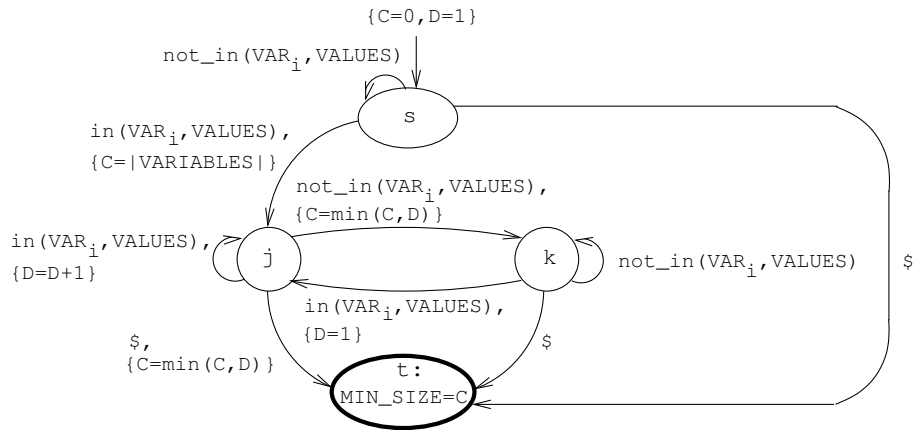


Figure 4.206: Automaton for the MIN_SIZE parameter of the group constraint

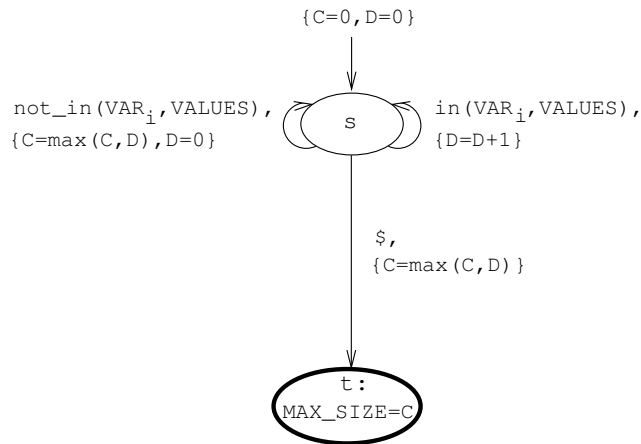


Figure 4.207: Automaton for the MAX_SIZE parameter of the group constraint

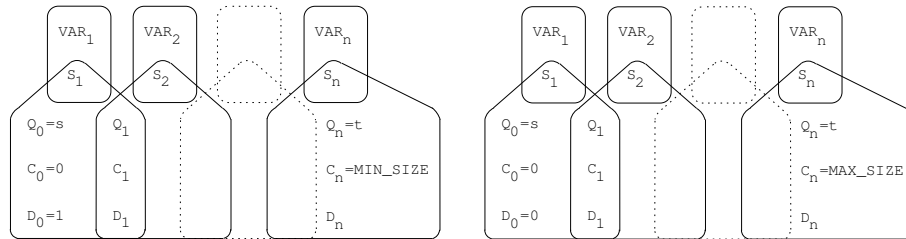


Figure 4.208: Hypergraphs of the reformulations corresponding to the automata of the MIN_SIZE and MAX_SIZE parameters of the group constraint

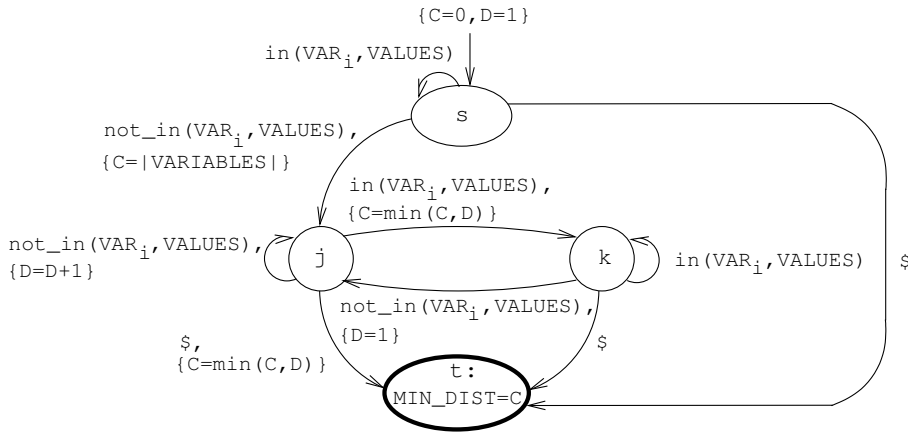


Figure 4.209: Automaton for the MIN_DIST parameter of the group constraint

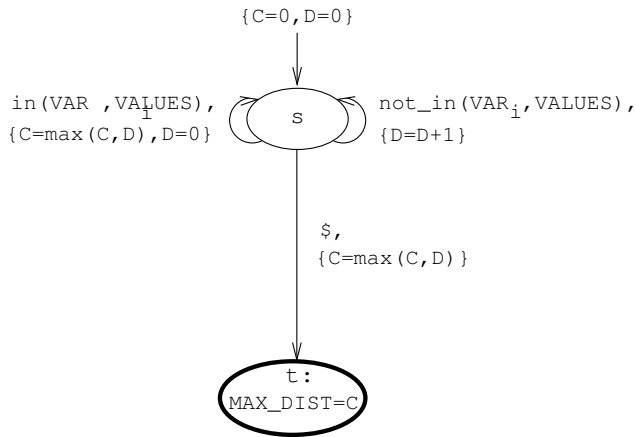


Figure 4.210: Automaton for the MAX_DIST parameter of the group constraint

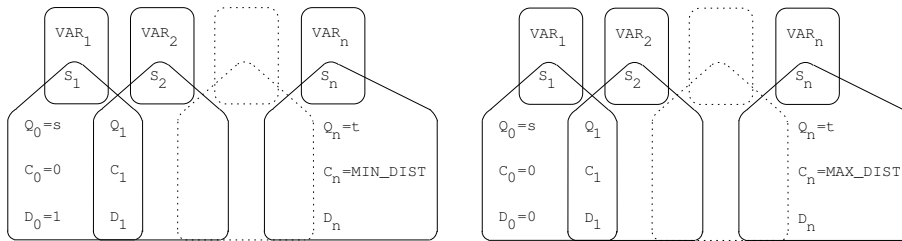


Figure 4.211: Hypergraphs of the reformulations corresponding to the automata of the MIN_DIST and MAX_DIST parameters of the group constraint

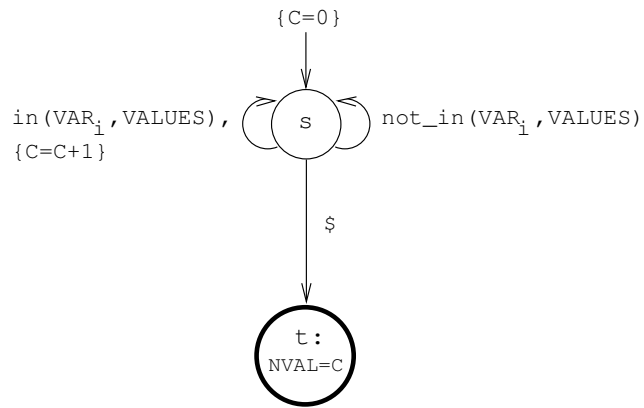


Figure 4.212: Automaton for the NVAL parameter of the group constraint

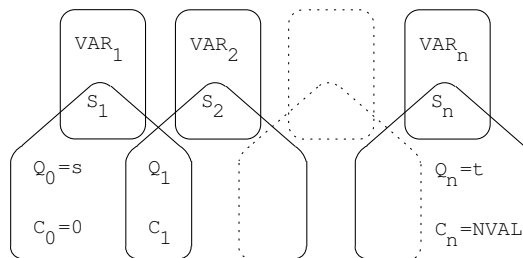


Figure 4.213: Hypergraph of the reformulation corresponding to the automaton of the NVAL parameter of the group constraint

work is represented by the variables of the VARIABLES collection. In this context the group constraint expresses for a person:

- The number of periods of consecutive night shift during a complete period of work.
- The total number of night shift during a complete period of work.
- The maximum number of allowed consecutive night shift.
- The minimum number of days (which do not correspond to night shift) between two consecutive sequences of night shift.

Remark

For this constraint we use the possibility to express directly more than one constraint on the characteristics of the final graph we want to obtain. For more propagation, it is crucial to keep this in one single constraint, since strong relations relate the different characteristics of a graph. This constraint is very similar to the group constraint introduced in CHIP, except that here, the MIN_DIST and MAX_DIST constraints apply also for the two borders: we cannot start or end with a group of k consecutive variables that take their values outside VALUES and such that k is less than MIN_DIST or k is greater than MAX_DIST.

See also

group_skip_isolated_item, change_continuity, stretch_path.

Key words

timetabling constraint, connected component, automaton, automaton with counters, alpha-acyclic constraint network(2), alpha-acyclic constraint network(3), vpartition, consecutive loops are connected.

4.98 `group_skip_isolated_item`

Origin	Derived from <code>group</code> .
Constraint	<code>group_skip_isolated_item(NGROUP, MIN_SIZE, MAX_SIZE, NVAL, VARIABLES, VALUES)</code>
Argument(s)	<pre> NGROUP : dvar MIN_SIZE : dvar MAX_SIZE : dvar NVAL : dvar VARIABLES : collection(var – dvar) VALUES : collection(val – int) </pre>

Restriction(s)	<pre> NGROUP ≥ 0 MIN_SIZE ≥ 0 MAX_SIZE ≥ MIN_SIZE NVAL ≥ 0 required(VARIABLES, var) required(VALUES, val) distinct(VALUES, val) </pre>
-----------------------	--

Purpose

Let n be the number of variables of the collection `VARIABLES`. Let X_i, X_{i+1}, \dots, X_j ($1 \leq i < j \leq n$) be consecutive variables of the collection of variables `VARIABLES` such that the following conditions apply:

- All variables X_i, \dots, X_j take their value in the set of values `VALUES`,
- $i = 1$ or X_{i-1} does not take a value in `VALUES`,
- $j = n$ or X_{j+1} does not take a value in `VALUES`.

We call such a set of variables a *group*. The constraint `group_skip_isolated_item` is true if all the following conditions hold:

- There are exactly `NGROUP` groups of variables,
- The number of variables of the smallest group is `MIN_SIZE`,
- The number of variables of the largest group is `MAX_SIZE`,
- The number of variables that take their value in the set of values `VALUES` is equal to `NVAL`.

Arc input(s)	<code>VARIABLES</code>
Arc generator	<code>CHAIN</code> \mapsto <code>collection(variables1, variables2)</code>
Arc arity	2
Arc constraint(s)	<ul style="list-style-type: none"> • <code>in(variables1.var, VALUES)</code> • <code>in(variables2.var, VALUES)</code>

- Graph property(ies)**
- NSCC = NGROUP
 - MIN_NSCC = MIN_SIZE
 - MAX_NSCC = MAX_SIZE
 - NVERTEX = NVAL
-

Example

$$\text{group_skip_isolated_item} \left(\begin{array}{c} 1, 2, 2, 3, \left\{ \begin{array}{l} \text{var} - 2, \\ \text{var} - 8, \\ \text{var} - 1, \\ \text{var} - 7, \\ \text{var} - 4, \\ \text{var} - 5, \\ \text{var} - 1, \\ \text{var} - 1, \\ \text{var} - 1 \end{array} \right\}, \\ \left\{ \begin{array}{l} \text{val} - 0, \\ \text{val} - 2, \\ \text{val} - 4, \\ \text{val} - 6, \\ \text{val} - 8 \end{array} \right\} \end{array} \right)$$

The previous constraint holds since:

- The final graph contains one strongly connected component. Therefore the number of groups is equal to one.
- The unique strongly connected component of the final graph contains two vertices. Therefore MIN_SIZE and MAX_SIZE are both equal to two.
- The number of vertices of the final graph is equal to two. Therefore NVAL is equal to two.

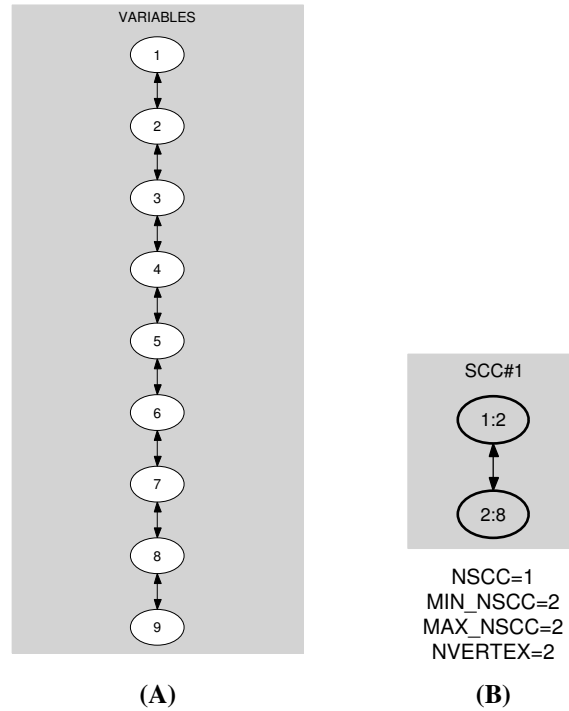
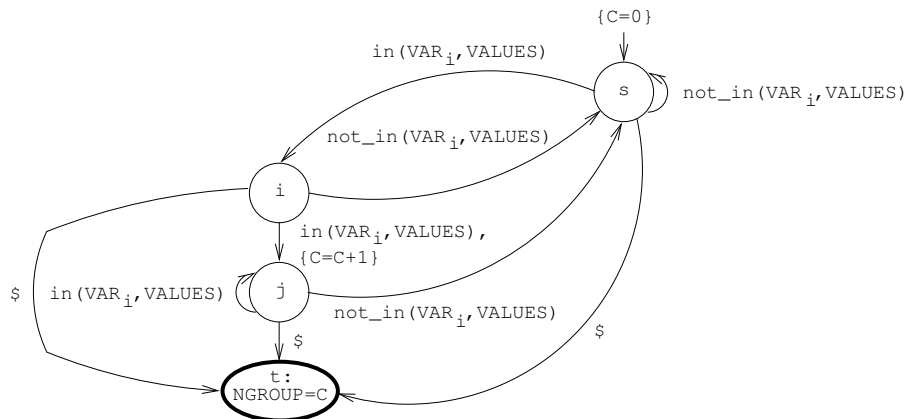
Parts (A) and (B) of Figure 4.214 respectively show the initial and final graph.

Graph model We use the *CHAIN* arc generator in order to produce the initial graph. This creates the graph depicted in part (A) of Figure 4.214. We use *CHAIN* together with the arc constraint $\text{variables1.var} \in \text{VALUES} \wedge \text{variables2.var} \in \text{VALUES}$ in order to skip the isolated variables that take a value in *VALUES* that we don't want to count as a group. This is why, on the example, value 4 is not counted as a group.

Automaton Figures 4.215, 4.217, 4.218 and 4.220 depict the different automata associated to the *group_skip_isolated_item* constraint. For the automata that respectively compute NGROUP, MIN_SIZE, MAX_SIZE and NVAL we have a 0-1 signature variable S_i for each variable VAR_i of the collection *VARIABLES*. The following signature constraint links VAR_i and S_i : $\text{VAR}_i \in \text{VALUES} \Leftrightarrow S_i$.

Usage This constraint is useful in order to specify rules about how rest days should be allocated to a person during a period of n consecutive days. In this case *VALUES* are the codes for the rest days (perhaps one single value) and *VARIABLES* corresponds to the amount of work done during n consecutive days. We can then express a rule like: In a month one should have at least 4 periods of at least 2 rest days; Isolated rest days are not counted as rest periods.

See also *group*, *change_continuity*, *stretch_path*.

Figure 4.214: Initial and final graph of the `group_skip_isolated_item` constraintFigure 4.215: Automaton for the `NGROUP` parameter of the `group_skip_isolated_item` constraint

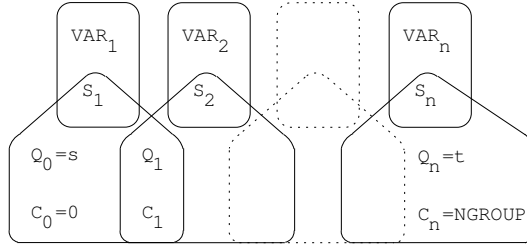


Figure 4.216: Hypergraph of the reformulation corresponding to the automaton of the NGROUP parameter of the group_skip_isolated_item constraint

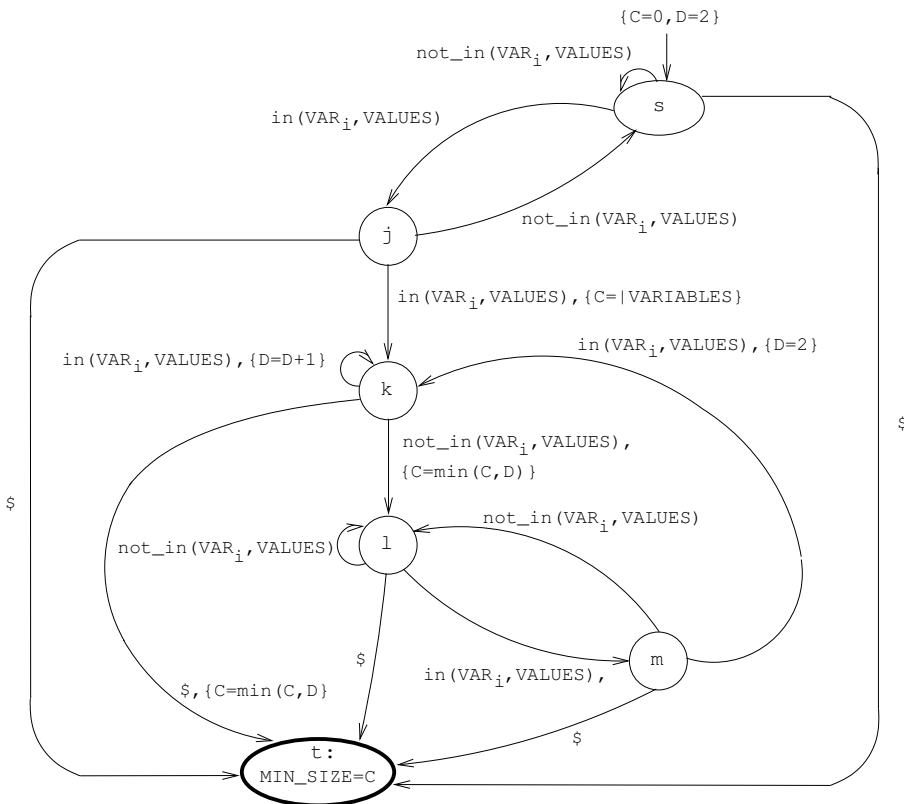


Figure 4.217: Automaton for the MIN_SIZE parameter of the group_skip_isolated_item constraint

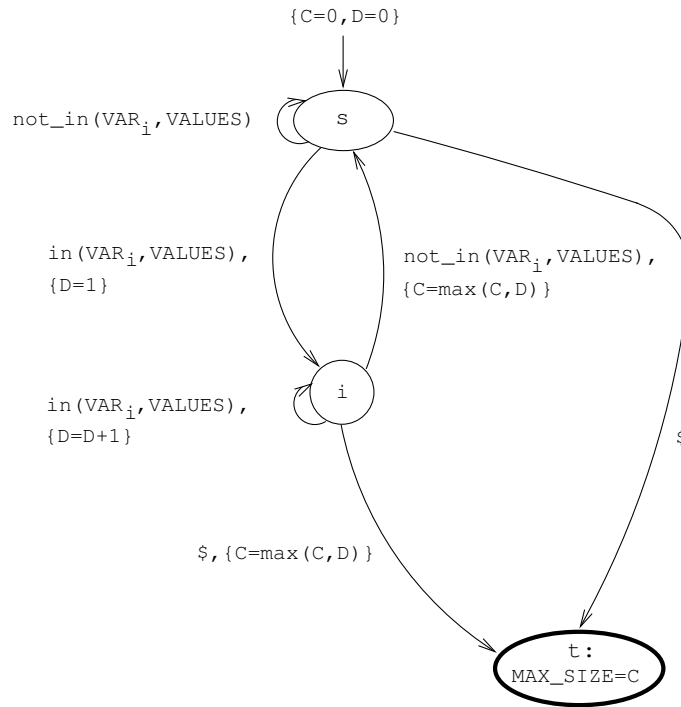


Figure 4.218: Automaton for the MAX_SIZE parameter of the group_skip_isolated_item constraint

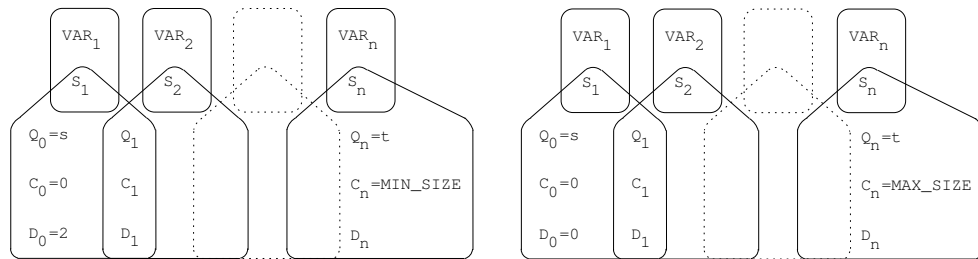


Figure 4.219: Hypergraphs of the reformulations corresponding to the automata of the MIN_SIZE and MAX_SIZE parameters of the group_skip_isolated_item constraint

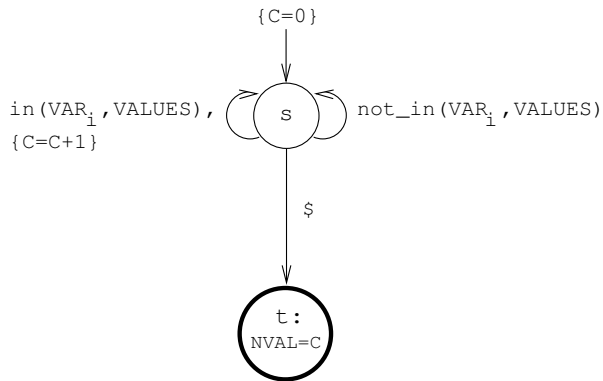


Figure 4.220: Automaton for the NVAL parameter of the group_skip_isolated_item constraint

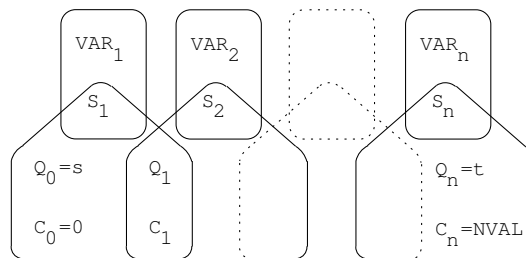


Figure 4.221: Hypergraph of the reformulation corresponding to the automaton of the NVAL parameter of the group_skip_isolated_item constraint

Key words

timetabling constraint, strongly connected component, automaton,
automaton with counters, alpha-acyclic constraint network(2),
alpha-acyclic constraint network(3).

20000128

531

4.99 highest_peak

Origin	Derived from peak.
Constraint	highest_peak(HEIGHT, VARIABLES)
Argument(s)	HEIGHT : dvar VARIABLES : collection(var - dvar)
Restriction(s)	HEIGHT \geq 0 VARIABLES.var \geq 0 required(VARIABLES, var)

Purpose

A variable V_k ($1 < k < m$) of the sequence of variables $VARIABLES = V_1, \dots, V_m$ is a *peak* if and only if there exist an i ($1 < i \leq k$) such that $V_{i-1} < V_i$ and $V_i = V_{i+1} = \dots = V_k$ and $V_k > V_{k+1}$. HEIGHT is the maximum value of the peak variables. If no such variable exists HEIGHT is equal to 0.

Example

$$\text{highest_peak} \left(8, \left\{ \begin{array}{l} \text{var} - 1, \\ \text{var} - 1, \\ \text{var} - 4, \\ \text{var} - 8, \\ \text{var} - 6, \\ \text{var} - 2, \\ \text{var} - 7, \\ \text{var} - 1 \end{array} \right\} \right)$$

The previous constraint holds since 8 is the maximum peak of the sequence 1 1 4 8 6 2 7 1.

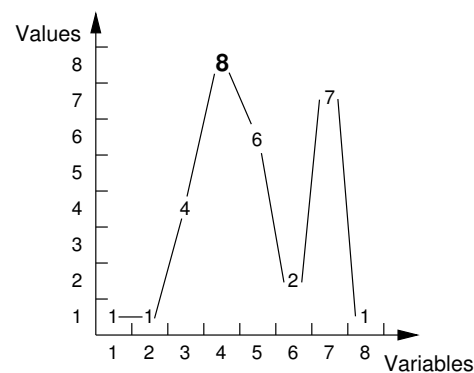


Figure 4.222: The sequence and its highest peak

Automaton

Figure 4.223 depicts the automaton associated to the highest_peak constraint. To each pair of consecutive variables (VAR_i, VAR_{i+1}) of the collection VARIABLES corresponds a signature variable S_i . The following signature constraint links VAR_i, VAR_{i+1} and S_i :

$$VAR_i > VAR_{i+1} \Leftrightarrow S_i = 0 \wedge VAR_i = VAR_{i+1} \Leftrightarrow S_i = 1 \wedge VAR_i < VAR_{i+1} \Leftrightarrow S_i = 2.$$

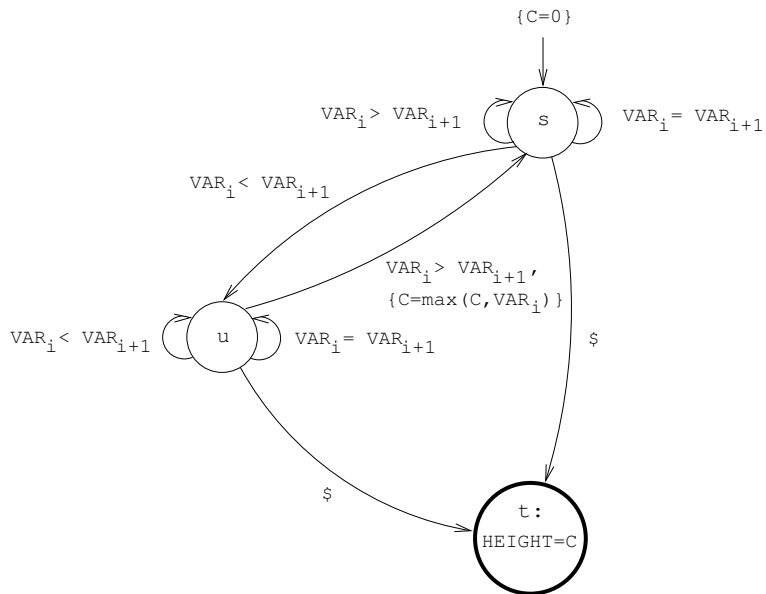


Figure 4.223: Automaton of the highest_peak constraint

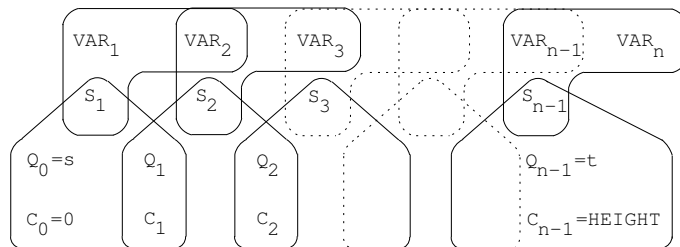


Figure 4.224: Hypergraph of the reformulation corresponding to the automaton of the highest_peak constraint

See also peak, deepest_valley.

Key words sequence, automaton, automaton with counters, sliding cyclic(1) constraint network(2).

20040530

535

4.100 in

Origin	Domain definition.
Constraint	$\text{in}(\text{VAR}, \text{VALUES})$
Argument(s)	VAR : dvar VALUES : collection(val – int)
Restriction(s)	required(VALUES, val) distinct(VALUES, val)
Purpose	Enforce the domain variable VAR to take a value within the values described by the VALUES collection.
Derived Collection(s)	$\text{col}(\text{VARIABLES} - \text{collection}(\text{var} - \text{dvar}), [\text{item}(\text{var} - \text{VAR})])$
Arc input(s)	VARIABLES VALUES
Arc generator	$\text{PRODUCT} \mapsto \text{collection}(\text{variables}, \text{values})$
Arc arity	2
Arc constraint(s)	variables.var = values.val
Graph property(ies)	$\text{NARC} = 1$
Example	$\text{in}(3, \{\text{val} - 1, \text{val} - 3\})$

Parts (A) and (B) of Figure 4.225 respectively show the initial and final graph. Since we use the **NARC** graph property, the unique arc of the final graph is stressed in bold.

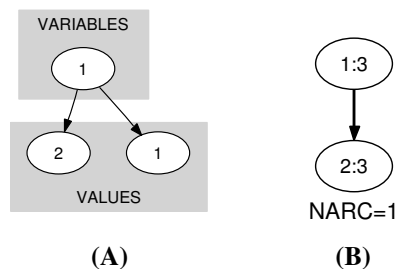


Figure 4.225: Initial and final graph of the **in** constraint

Signature

Since all the val attributes of the VALUES collection are distinct and because of the arc constraint variables.var = values.val the final graph contains at most one arc. Therefore we can rewrite $\text{NARC} = 1$ to $\text{NARC} \geq 1$ and simplify NARC to NARC.

Automaton

Figure 4.226 depicts the automaton associated to the `in` constraint. Let VAL_i be the `val` attribute of the i^{th} item of the `VALUES` collection. To each pair (VAR, VAL_i) corresponds a 0-1 signature variable S_i as well as the following signature constraint: $VAR = VAL_i \Leftrightarrow S_i$.

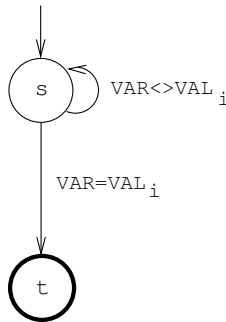


Figure 4.226: Automaton of the `in` constraint

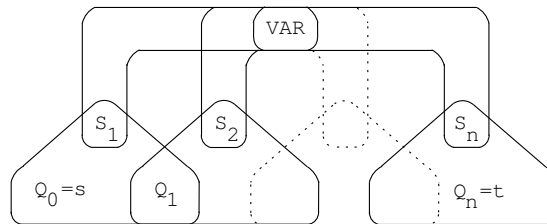


Figure 4.227: Hypergraph of the reformulation corresponding to the automaton of the `in` constraint

Remark

Entailment occurs immediately after posting this constraint.

Used in

`among`, `cardinality_atmost_partition`, `group`, `group_skip_isolated_item`, `in_same_partition`.

See also

`not_in`, `in_same_partition`.

Key words

value constraint, unary constraint, included, domain definition, automaton, automaton without counters, centered cyclic(1) constraint network(1), derived collection.

4.101 in_relation

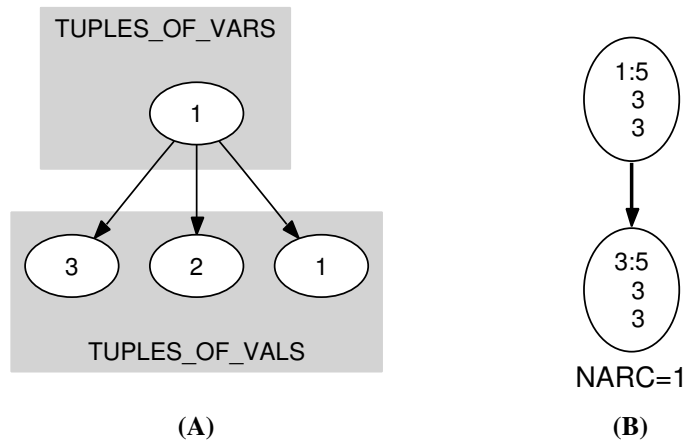
Origin	Constraint explicitly defined by tuples of values.
Constraint	<code>in_relation(VARIABLES, TUPLES_OF_VALS)</code>
Synonym(s)	<code>extension.</code>
Type(s)	TUPLE_OF_VARS : <code>collection(var - dvar)</code> TUPLE_OF_VALS : <code>collection(val - int)</code>
Argument(s)	VARIABLES : TUPLE_OF_VARS TUPLES_OF_VALS : <code>collection(tuple - TUPLE_OF_VALS)</code>
Restriction(s)	<code>required(TUPLE_OF_VARS, var)</code> <code>required(TUPLE_OF_VALS, val)</code> <code>required(TUPLES_OF_VALS, tuple)</code> <code>min_size(TUPLES_OF_VALS, tuple) = VARIABLES </code> <code>max_size(TUPLES_OF_VALS, tuple) = VARIABLES </code>
Purpose	Enforce the tuple of variables <code>VARIABLES</code> to take its value out of a set of tuples of values <code>TUPLES_OF_VALS</code> . The <i>value</i> of a tuple of variables $\langle V_1, V_2, \dots, V_n \rangle$ is a tuple of values $\langle U_1, U_2, \dots, U_n \rangle$ if and only if $V_1 = U_1 \wedge V_2 = U_2 \wedge \dots \wedge V_n = U_n$.
Derived Collection(s)	<code>col(TUPLES_OF_VARS - collection(vec - TUPLE_OF_VARS), [item(vec - VARIABLE)])</code>
Arc input(s)	TUPLES_OF_VARS TUPLES_OF_VALS
Arc generator	$\text{PRODUCT} \mapsto \text{collection}(\text{tuples_of_vars}, \text{tuples_of_vals})$
Arc arity	2
Arc constraint(s)	<code>vec.eq_tuple(tuples_of_vars.vec, tuples_of_vals.tuple)</code>
Graph property(ies)	$\text{NARC} \geq 1$

Example `in_relation` $\left(\begin{array}{l} \{ \text{var} - 5, \text{var} - 3, \text{var} - 3 \}, \\ \left\{ \begin{array}{l} \text{tuple} - \{ \text{val} - 5, \text{val} - 2, \text{val} - 3 \}, \\ \text{tuple} - \{ \text{val} - 5, \text{val} - 2, \text{val} - 6 \}, \\ \text{tuple} - \{ \text{val} - 5, \text{val} - 3, \text{val} - 3 \} \end{array} \right\} \end{array} \right)$

Parts (A) and (B) of Figure 4.228 respectively show the initial and final graph. Since we use the **NARC** graph property, the unique arc of the final graph is stressed in bold.

Usage Quite often some constraints cannot be easily expressed, neither by a formula, nor by a regular pattern. In this case one has to define the constraint by specifying in extension the combinations of allowed values.

- Remark** Within [34] this constraint is called *extension*.
- See also** *element*.
- Key words** data constraint, tuple, extension, relation, derived collection.

Figure 4.228: Initial and final graph of the `in_relation` constraint

20030820

541

4.102 in_same_partition

Origin	Used for defining several entries of this catalog.
Constraint	<code>in_same_partition(VAR1, VAR2, PARTITIONS)</code>
Type(s)	<code>VALUES : collection(val – int)</code>
Argument(s)	<pre> VAR1 : dvar VAR2 : dvar PARTITIONS : collection(p – VALUES) </pre>
Restriction(s)	<pre> required(VALUES, val) distinct(VALUES, val) required(PARTITIONS, p) PARTITIONS ≥ 2 </pre>
Purpose	<p>Enforce VAR1 and VAR2 to be respectively assigned to values v_1 and v_2 that both belong to a same partition of the collection PARTITIONS.</p>
Derived Collection(s)	<code>col(VARIABLES – collection(var – dvar), [item(var – VAR1), item(var – VAR2)])</code>
Arc input(s)	VARIABLES PARTITIONS
Arc generator	$PRODUCT \mapsto \text{collection}(\text{variables}, \text{partitions})$
Arc arity	2
Arc constraint(s)	<code>in(variables.var, partitions.p)</code>
Graph property(ies)	<ul style="list-style-type: none"> • NSOURCE = 2 • NSINK = 1
Example	$\text{in_same_partition} \left(6, 2, \left\{ \begin{array}{l} p - \{\text{val} - 1, \text{val} - 3\}, \\ p - \{\text{val} - 4\}, \\ p - \{\text{val} - 2, \text{val} - 6\} \end{array} \right\} \right)$ <p>Parts (A) and (B) of Figure 4.229 respectively show the initial and final graph. Since we both use the NSOURCE and NSINK graph properties, the source and sink vertices of the final graph are shown with a double circle.</p>
Graph model	<p>VAR1 and VAR2 are put together in the derived collection VARIABLES. Since both VAR1 and VAR2 should take their value in one of the partition depicted by the PARTITIONS collection, the final graph should have two sources corresponding respectively to VAR1 and VAR2. Since two, possibly distinct, values should be assigned to VAR1 and VAR2 and since these values belong to the same partition p the final graph should only have one sink. This sink corresponds in fact to partition p.</p>

Signature	Observe that the sinks of the initial graph cannot become sources of the final graph since isolated vertices are eliminated from the final graph. Since the final graph contains two sources it also includes one arc between a source and a sink. Therefore the minimum number of sinks of the final graph is equal to one. So we can rewrite $NSINK = 1$ to $NSINK \geq 1$ and simplify \overline{NSINK} to \overline{NSINK} .
Automaton	Figure 4.230 depicts the automaton associated to the <code>in_same_partition</code> constraint. Let $VALUES_i$ be the <code>p</code> attribute of the i^{th} item of the <code>PARTITIONS</code> collection. To each triple $(VAR1, VAR2, VALUES_i)$ corresponds a 0-1 signature variable S_i as well as the following signature constraint: $((VAR1 \in VALUES_i) \wedge (VAR2 \in VALUES_i)) \Leftrightarrow S_i$.
Used in	<code>alldifferent_partition</code> , <code>balance_partition</code> , <code>change_partition</code> , <code>common_partition</code> , <code>nclass</code> , <code>same_partition</code> , <code>soft_same_partition_var</code> , <code>soft_used_by_partition_var</code> , <code>used_by_partition</code> .
See also	<code>in</code> .
Key words	value constraint, partition, automaton, automaton without counters, centered cyclic(2) constraint network(1), derived collection.

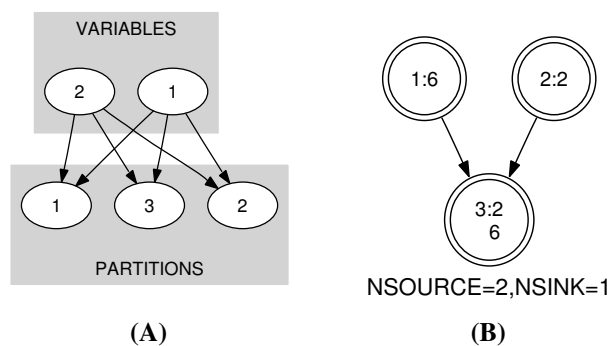
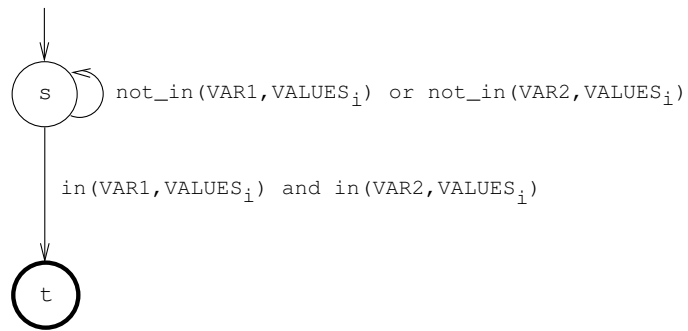
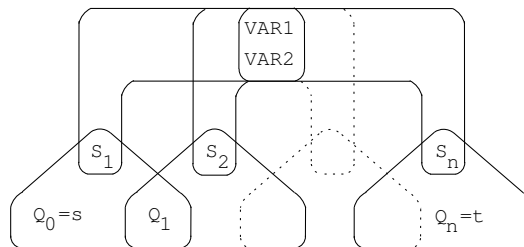


Figure 4.229: Initial and final graph of the `in_same_partition` constraint

Figure 4.230: Automaton of the `in_same_partition` constraintFigure 4.231: Hypergraph of the reformulation corresponding to the automaton of the `in_same_partition` constraint

20030820

545

4.103 `in_set`

Origin	Used for defining constraints with set variables.
Constraint	<code>in_set(VAL, SET)</code>
Argument(s)	VAL : dvar SET : svar
Purpose	Constraint variable VAL to belong to set SET.
Example	<code>in_set(3, {1, 3})</code>
Used in	<code>clique</code> , <code>cutset</code> , <code>discrepancy</code> , <code>inverse_set</code> , <code>k_cut</code> , <code>link_set_to_booleans</code> , <code>path_from_to</code> , <code>strongly_connected</code> , <code>sum</code> , <code>sum_set</code> , <code>symmetric_cardinality</code> , <code>symmetric_gcc</code> , <code>tour</code> .
Key words	predefined constraint, value constraint, included, constraint involving set variables.

20030820

547

4.104 increasing

Origin	KOALOG
Constraint	increasing(VARIABLES)
Argument(s)	VARIABLES : collection(var - dvar)
Restriction(s)	$ \text{VARIABLES} > 0$ required(VARIABLES, var)
Purpose	The variables of the collection VARIABLES are increasing.
Arc input(s)	VARIABLES
Arc generator	$\text{PATH} \mapsto \text{collection}(\text{variables1}, \text{variables2})$
Arc arity	2
Arc constraint(s)	$\text{variables1.var} \leq \text{variables2.var}$
Graph property(ies)	$\text{NARC} = \text{VARIABLES} - 1$
Example	increasing({var - 1, var - 1, var - 4, var - 8})

Parts (A) and (B) of Figure 4.232 respectively show the initial and final graph. Since we use the **NARC** graph property, the arcs of the final graph are stressed in bold.

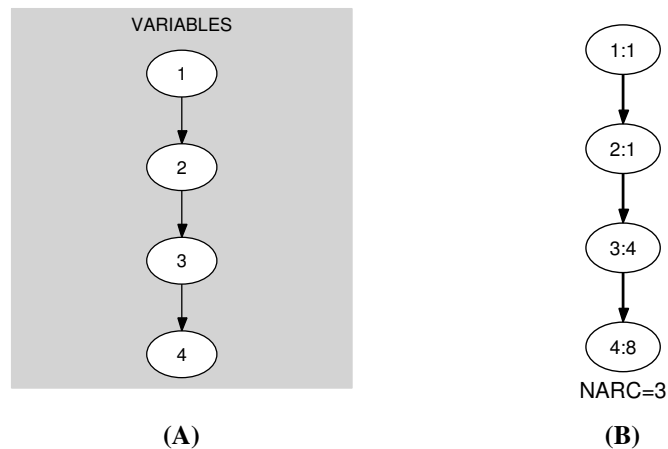


Figure 4.232: Initial and final graph of the increasing constraint

- Automaton** Figure 4.233 depicts the automaton associated to the `increasing` constraint. To each pair of consecutive variables (VAR_i, VAR_{i+1}) of the collection `VARIABLES` corresponds a 0-1 signature variable S_i . The following signature constraint links VAR_i , VAR_{i+1} and S_i : $VAR_i > VAR_{i+1} \Leftrightarrow S_i$.
- See also** `strictly_increasing`, `decreasing`, `strictly_decreasing`.
- Key words** decomposition, order constraint, automaton, automaton without counters, sliding cyclic(1) constraint network(1).

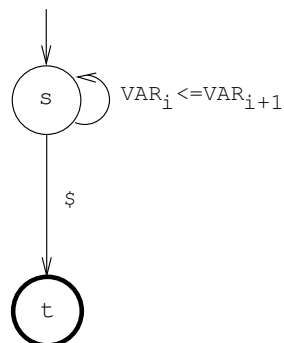


Figure 4.233: Automaton of the increasing constraint

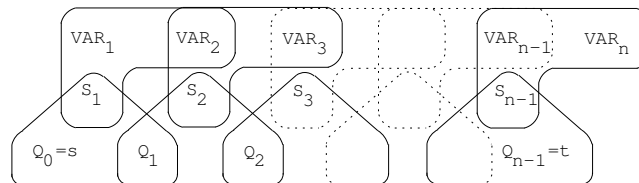


Figure 4.234: Hypergraph of the reformulation corresponding to the automaton of the increasing constraint

20040814

551

4.105 indexed_sum

Origin	N. Beldiceanu
Constraint	indexed_sum(ITEMS, TABLE)
Argument(s)	ITEMS : collection(index - dvar, weight - dvar) TABLE : collection(index - int, sum - dvar)
Restriction(s)	ITEMS > 0 TABLE > 0 required(ITEMS, [index, weight]) ITEMS.index ≥ 0 ITEMS.index < TABLE required(TABLE, [index, sum]) TABLE.index ≥ 0 TABLE.index < TABLE increasing_seq(TABLE, index)

Purpose

Given several items of the collection ITEMS (each of them having a specific fixed index as well as a weight which may be negative or positive), and a table TABLE (each entry of TABLE corresponding to a sum variable), assign each item to an entry of TABLE so that the sum of the weights of the items assigned to that entry is equal to the corresponding sum variable.

For all items of TABLE:

Arc input(s)	ITEMS TABLE
Arc generator	PRODUCT \mapsto collection(items, table)
Arc arity	2
Arc constraint(s)	items.index = table.index
Sets	SUCC \mapsto $\left[\begin{array}{l} \text{source,} \\ \text{variables - col} \left(\begin{array}{l} \text{VARIABLES - collection(var - dvar),} \\ \text{[item(var - ITEMS.weight)]} \end{array} \right) \end{array} \right]$
Constraint(s) on sets	sum_ctr(variables, =, TABLE.sum)

Example

$$\text{indexed_sum} \left(\left(\begin{array}{l} \left\{ \begin{array}{ll} \text{index - 2} & \text{weight - -4,} \\ \text{index - 0} & \text{weight - 6,} \end{array} \right\}, \\ \left\{ \begin{array}{ll} \text{index - 2} & \text{weight - 1} \\ \text{index - 0} & \text{sum - 6,} \\ \text{index - 1} & \text{sum - 0,} \\ \text{index - 2} & \text{sum - -3} \end{array} \right\} \end{array} \right) \right)$$

Part (A) of Figure 4.235 shows the initial graphs associated to entries 0, 1 and 2. Part (B) of Figure 4.235 shows the corresponding final graphs associated to entries 0 and 2. Each source vertex of the final graph can be interpreted as an item assigned to a specific entry of TABLE. The `indexed_sum` constraint holds since the sum variables associated to each entry of TABLE are equal to the sum of the weights of the items assigned to the corresponding entry.

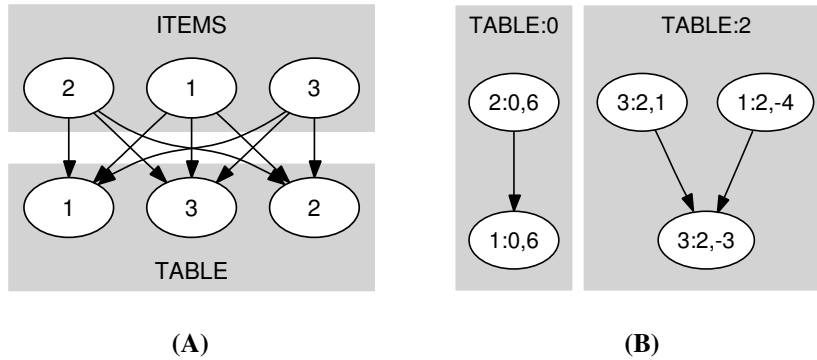


Figure 4.235: Initial and final graph of the `indexed_sum` constraint

Graph model

We enforce the `sum_ctr` constraint on the weight of the items that are assigned to the same entry.

See also

`bin_packing`.

Key words

assignment, variable indexing, variable subscript.

4.106 inflexion

Origin	N. Beldiceanu
Constraint	<code>inflexion(N, VARIABLES)</code>
Argument(s)	<code>N</code> : dvar <code>VARIABLES</code> : collection(var - dvar)
Restriction(s)	$N \geq 1$ $N \leq \text{VARIABLES} $ <code>required(VARIABLES, var)</code>

Purpose

N is equal to the number of times that the following conjunctions of constraints hold:

- $X_i \text{CTR} X_{i+1} \wedge X_i \neq X_{i+1}$,
- $X_{i+1} = X_{i+2} \wedge \dots \wedge X_{j-2} = X_{j-1}$,
- $X_{j-1} \neq X_j \wedge X_{j-1} \neg\text{CTR} X_j$.

where X_k is the k^{th} item of the `VARIABLES` collection and $1 \leq i, i+2 \leq j, j \leq n$ and `CTR` is `< or >`.

Example

$$\text{inflexion} \left(3, \left\{ \begin{array}{l} \text{var} - 1, \\ \text{var} - 1, \\ \text{var} - 4, \\ \text{var} - 8, \\ \text{var} - 8, \\ \text{var} - 2, \\ \text{var} - 7, \\ \text{var} - 1 \end{array} \right\} \right)$$

The previous constraint holds since the sequence 1 1 4 8 8 2 7 1 contains three inflexions peaks which respectively correspond to values 8, 2 and 7.

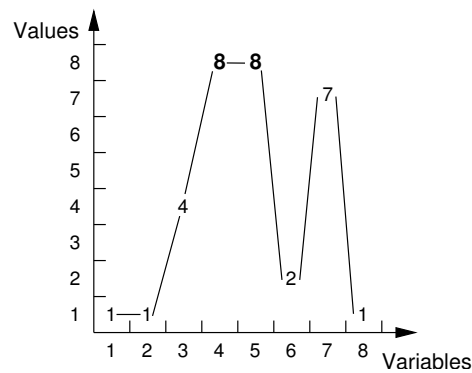


Figure 4.236: The sequence and its three inflexions

Automaton

Figure 4.237 depicts the automaton associated to the *inflexion* constraint. To each pair of consecutive variables (VAR_i, VAR_{i+1}) of the collection *VARIABLES* corresponds a signature variable S_i . The following signature constraint links VAR_i, VAR_{i+1} and S_i : $(VAR_i > VAR_{i+1} \Leftrightarrow S_i = 0) \wedge (VAR_i = VAR_{i+1} \Leftrightarrow S_i = 1) \wedge (VAR_i < VAR_{i+1} \Leftrightarrow S_i = 2)$.

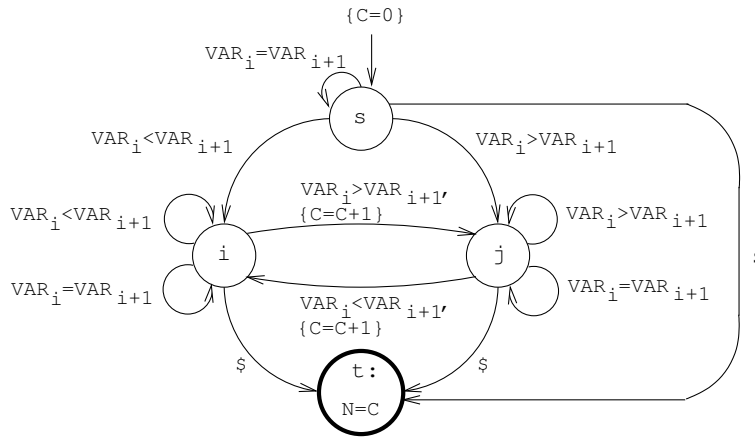


Figure 4.237: Automaton of the *inflexion* constraint

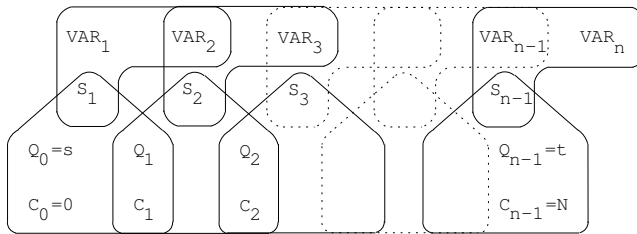


Figure 4.238: Hypergraph of the reformulation corresponding to the automaton of the *inflexion* constraint

- Usage** Useful for constraining the number of *inflexions* of a sequence of domain variables.
- Remark** Since the arity of the arc constraint is not fixed, the *inflexion* constraint cannot be currently described. However, this would not hold anymore if we were introducing a slot that specifies how to merge adjacent vertices of the final graph.
- See also** *peak, valley.*
- Key words** *sequence, automaton, automaton with counters, sliding cyclic(1) constraint network(2).*

4.107 int_value_precede

Origin	[121]
Constraint	int_value_precede(S, T, VARIABLES)
Argument(s)	S : int T : int VARIABLES : collection(var - dvar)
Restriction(s)	S ≠ T required(VARIABLES, var)
Purpose	If value T occurs in the collection of variables VARIABLES then its first occurrence should be preceded by an occurrence of value S.

Example int_value_precede $\left(0, 1, \left\{ \begin{array}{l} \text{var} - 4, \\ \text{var} - 0, \\ \text{var} - 6, \\ \text{var} - 1, \\ \text{var} - 0 \end{array} \right\} \right)$

The int_value_precede constraint holds since the first occurrence of value 0 precedes the first occurrence of value 1.

Automaton Figure 4.239 depicts the automaton associated to the int_value_precede constraint. Let VAR_{*i*} be the *i*th variable of the VARIABLES collection. To each triple (S, T, VAR_{*i*}) corresponds a signature variable S_{*i*} as well as the following signature constraint: (VAR_{*i*} = S ⇔ S_{*i*} = 1) ∧ (VAR_{*i*} = T ⇔ S_{*i*} = 2) ∧ (VAR_{*i*} ≠ S ∧ VAR_{*i*} ≠ T ⇔ S_{*i*} = 3).

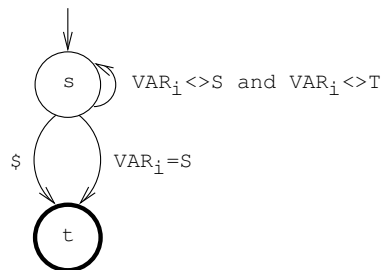


Figure 4.239: Automaton of the int_value_precede constraint

Algorithm A filtering algorithm for maintaining value precedence is presented in [121]. Its complexity is linear to the number of variables of the collection VARIABLES.

See also int_value_precede_chain, set_value_precede.

Key words order constraint, symmetry, indistinguishable values, value precedence, Berge-acyclic constraint network, automaton, automaton without counters.

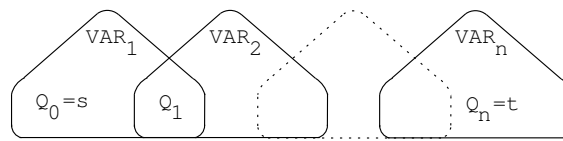


Figure 4.240: Hypergraph of the reformulation corresponding to the automaton of the `int_value_precede` constraint

4.108 int_value_precede_chain

Origin	[121]
Constraint	int_value_precede_chain(VALUEs, VARIABLEs)
Argument(s)	VALUEs : collection(val - int) VARIABLEs : collection(var - dvar)
Restriction(s)	required(VALUEs, val) distinct(VALUEs, val) required(VARIABLEs, var)

Purpose

Assuming n denotes the number of items of the VALUEs collection, the following condition holds for every $i \in [1, n - 1]$: When it exists, the first occurrence of the $(i + 1)^{th}$ value of the VALUEs collection should be preceded by the first occurrence of the i^{th} value of the VALUEs collection.

Example

$$\text{int_value_precede_chain} \left(\begin{array}{c} \{ \text{val} - 4, \text{val} - 0, \text{val} - 1 \}, \\ \left\{ \begin{array}{c} \text{var} - 4, \\ \text{var} - 0, \\ \text{var} - 6, \\ \text{var} - 1, \\ \text{var} - 0 \end{array} \right\} \end{array} \right)$$

The int_value_precede_chain constraint holds since:

- The first occurrence of value 4 occurs before the first occurrence of value 0.
- The first occurrence of value 0 occurs before the first occurrence of value 1.

Automaton

Figure 4.241 depicts the automaton associated to the int_value_precede_chain constraint. Let VAR_i be the i^{th} variable of the VARIABLEs collection. Let VAL_j ($1 < j < |\text{VALUEs}|$) denotes the j^{th} value of the VALUEs collection. To each variable VAR_i corresponds a signature variable S_i as well as the following signature constraint: $(\text{VAR}_i \notin \text{VALUEs} \Leftrightarrow S_i = 0) \wedge (\text{VAR}_i = \text{VAL}_1 \Leftrightarrow S_i = 1) \wedge (\text{VAR}_i = \text{VAL}_2 \Leftrightarrow S_i = 2) \wedge \dots \wedge (\text{VAR}_i = \text{VAL}_{|\text{VALUEs}|} \Leftrightarrow S_i = |\text{VALUEs}|)$.

Algorithm

The reformulation associated to the previous automaton achieves to arc-consistency.

See also

int_value_precede.

Key words

order constraint, symmetry, indistinguishable values, value precedence, Berge-acyclic constraint network, automaton, automaton without counters.

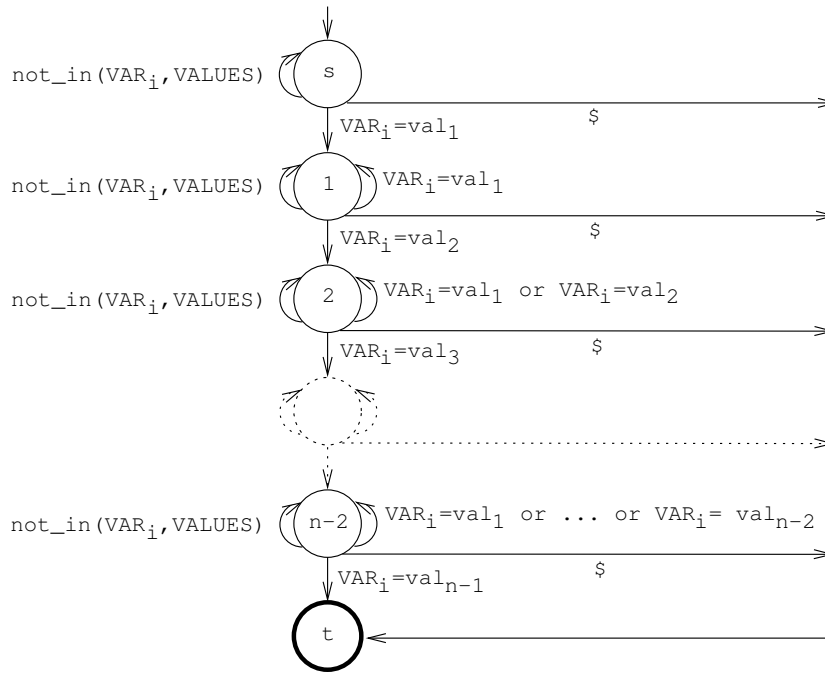


Figure 4.241: Automaton of the `int_value_precede_chain` constraint

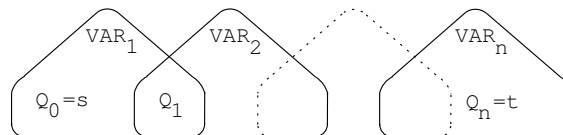


Figure 4.242: Hypergraph of the reformulation corresponding to the automaton of the `int_value_precede_chain` constraint

4.109 interval_and_count

Origin	[122]
Constraint	interval_and_count(ATMOST, COLOURS, TASKS, SIZE_INTERVAL)
Argument(s)	<pre> ATMOST : int COLOURS : collection(val - int) TASKS : collection(origin - dvar, colour - dvar) SIZE_INTERVAL : int </pre>
Restriction(s)	<pre> ATMOST ≥ 0 required(COLOURS, val) distinct(COLOURS, val) required(TASKS, [origin, colour]) SIZE_INTERVAL > 0 </pre>
Purpose	<p>First consider the set of tasks of the TASKS collection, where each task has a specific colour which may not be initially fixed. Then consider the intervals of the form $[k \cdot \text{SIZE_INTERVAL}, k \cdot \text{SIZE_INTERVAL} + \text{SIZE_INTERVAL} - 1]$, where k is an integer. The interval_and_count constraint enforces that, for each interval I_k previously defined, the total number of tasks which both are assigned to I_k and take their colour in COLOURS does not exceed the limit ATMOST.</p>
Arc input(s)	TASKS TASKS
Arc generator	$PRODUCT \mapsto \text{collection}(\text{tasks1}, \text{tasks2})$
Arc arity	2
Arc constraint(s)	$\text{tasks1.origin}/\text{SIZE_INTERVAL} = \text{tasks2.origin}/\text{SIZE_INTERVAL}$
Sets	$\text{SUCC} \mapsto \left[\begin{array}{l} \text{source}, \\ \text{variables} - \text{col} \left(\begin{array}{l} \text{VARIABLES} - \text{collection}(\text{var} - \text{dvar}), \\ [\text{item}(\text{var} - \text{TASKS.colour})] \end{array} \right) \end{array} \right]$
Constraint(s) on sets	among_low_up(0, ATMOST, variables, COLOURS)

Example	$\text{interval_and_count} \left(\begin{array}{l} 2, \{\text{val} - 4\}, \\ \left\{ \begin{array}{ll} \text{origin} - 1 & \text{colour} - 4, \\ \text{origin} - 0 & \text{colour} - 9, \\ \text{origin} - 10 & \text{colour} - 4, \\ \text{origin} - 4 & \text{colour} - 4 \end{array} \right\}, 5 \end{array} \right)$
----------------	--

Figure 4.243 shows the solution associated to the previous example. The constraint interval_and_count holds since, for each interval, the number of tasks taking colour 4 does not exceed the limit 2. Parts (A) and (B) of Figure 4.244 respectively show the initial and final graph. Each connected component of the final graph corresponds to items which are all assigned to the same interval.

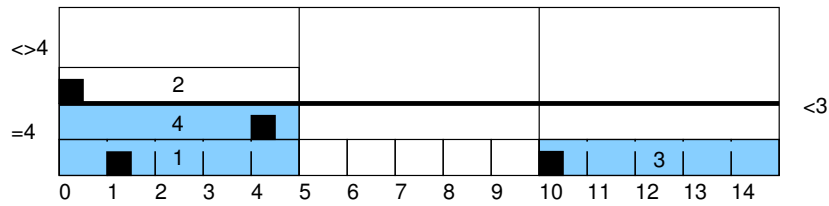


Figure 4.243: Solution with the use of each interval

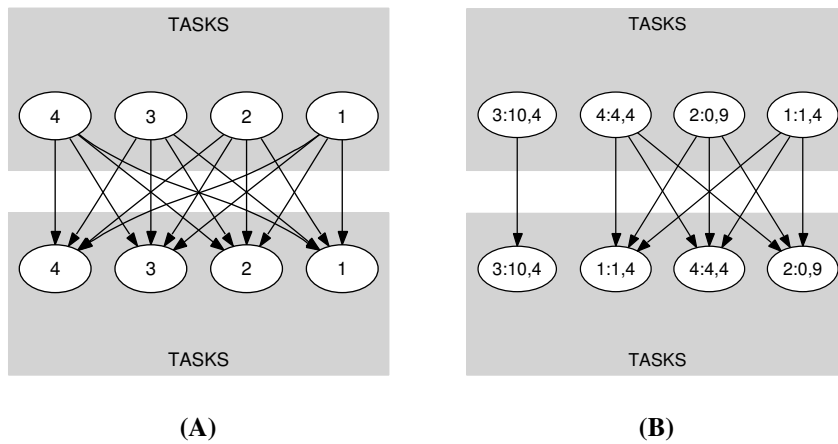


Figure 4.244: Initial and final graph of the interval_and_count constraint

Graph model

We use a bipartite graph where each class of vertices corresponds to the different tasks of the TASKS collection. There is an arc between two tasks if their origins belong to the same interval. Finally we enforce an `among_low_up` constraint on each set S of successors of the different vertices of the final graph. This put a restriction on the maximum number of tasks of S for which the colour attribute takes its value in COLOURS.

Automaton

Figure 4.245 depicts the automaton associated to the `interval_and_count` constraint. Let $COLOUR_i$ be the `colour` attribute of the i^{th} item of the TASKS collection. To each pair $(COLOURS, COLOUR_i)$ corresponds a signature variable S_i as well as the following signature constraint: $COLOUR_i \in COLOURS \Leftrightarrow S_i$.

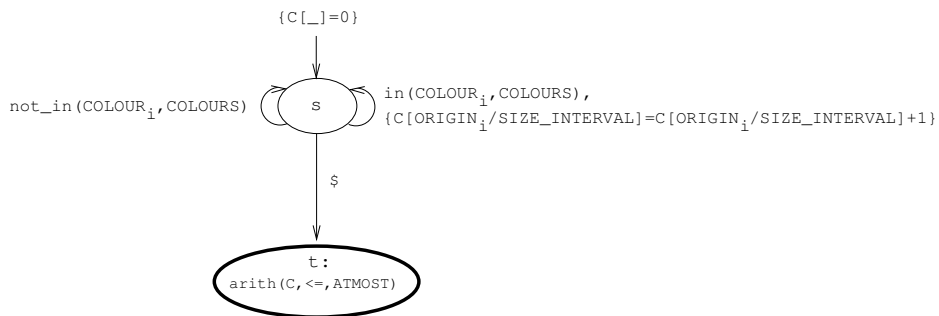


Figure 4.245: Automaton of the `interval_and_count` constraint

Usage

This constraint was originally proposed for dealing with timetabling problems. In this context the different intervals are interpreted as morning and afternoon periods of different consecutive days. Each colour corresponds to a type of course (i.e. French, mathematics). There is a restriction on the maximum number of courses of a given type each morning as well as each afternoon.

Remark

If we want to only consider intervals that correspond to the morning or to the afternoon we could extend the `interval_and_count` constraint in the following way:

- We introduce two extra parameters `REST` and `QUOTIENT` that correspond to non-negative integers such that `REST` is strictly less than `QUOTIENT`,
- We add the following condition to the arc constraint: $(tasks1.origin/SIZE_INTERVAL) \equiv REST \pmod{QUOTIENT}$

Now, if we want to express a constraint on the morning intervals, we set `REST` to 0 and `QUOTIENT` to 2.

See also

`count`, `among_low_up`.

Key words

timetabling constraint, resource constraint, temporal constraint, assignment, interval, coloured, automaton, automaton with array of counters.

20000128

563

4.110 interval_and_sum

Origin	Derived from cumulative.
Constraint	interval_and_sum(SIZE_INTERVAL, TASKS, LIMIT)
Argument(s)	SIZE_INTERVAL : int TASKS : collection(origin - dvar, height - dvar) LIMIT : int
Restriction(s)	SIZE_INTERVAL > 0 required(TASKS, [origin, height]) TASKS.height ≥ 0 LIMIT ≥ 0
Purpose	A maximum resource capacity constraint: We have to fix the origins of a collection of tasks in such a way that, for all the tasks that are allocated to the same interval, the sum of the heights does not exceed a given capacity. All the intervals we consider have the following form: $[k \cdot \text{SIZE_INTERVAL}, k \cdot \text{SIZE_INTERVAL} + \text{SIZE_INTERVAL} - 1]$, where k is an integer.
Arc input(s)	TASKS TASKS
Arc generator	$PRODUCT \mapsto \text{collection}(\text{tasks1}, \text{tasks2})$
Arc arity	2
Arc constraint(s)	$\text{tasks1.origin}/\text{SIZE_INTERVAL} = \text{tasks2.origin}/\text{SIZE_INTERVAL}$
Sets	$SUCC \mapsto$ $\left[\begin{array}{l} \text{source}, \\ \text{variables} - \text{col} \left(\begin{array}{l} \text{VARIABLES} - \text{collection}(\text{var} - \text{dvar}), \\ [\text{item}(\text{var} - \text{TASKS.height})] \end{array} \right) \end{array} \right]$
Constraint(s) on sets	$\text{sum_ctr}(\text{variables}, \leq, \text{LIMIT})$
Example	$\text{interval_and_sum} \left(5, \left\{ \begin{array}{ll} \text{origin} - 1 & \text{height} - 2, \\ \text{origin} - 10 & \text{height} - 2, \\ \text{origin} - 10 & \text{height} - 3, \\ \text{origin} - 4 & \text{height} - 1 \end{array} \right\}, 5 \right)$

Figure 4.246 shows the solution associated to the previous example. The constraint `interval_and_sum` holds since the sum of the heights of the tasks that are located in the same interval does not exceed the limit 5. Each task t is depicted by a rectangle r associated to the interval to which the task t is assigned. The rectangle r is labelled with the position of t within the items of the `TASKS` collection. The origin of task t is represented by a small black square located within its corresponding rectangle r . Finally, the height of a rectangle r is equal to the height of the task t to which it corresponds.

Parts (A) and (B) of Figure 4.247 respectively show the initial and final graph. Each connected component of the final graph corresponds to items which are all assigned to the same interval.

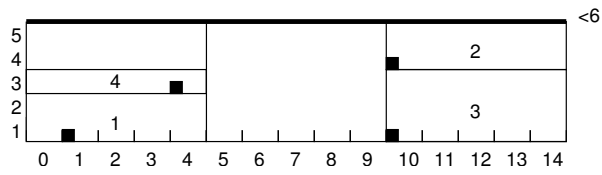


Figure 4.246: Solution showing for each interval the corresponding tasks

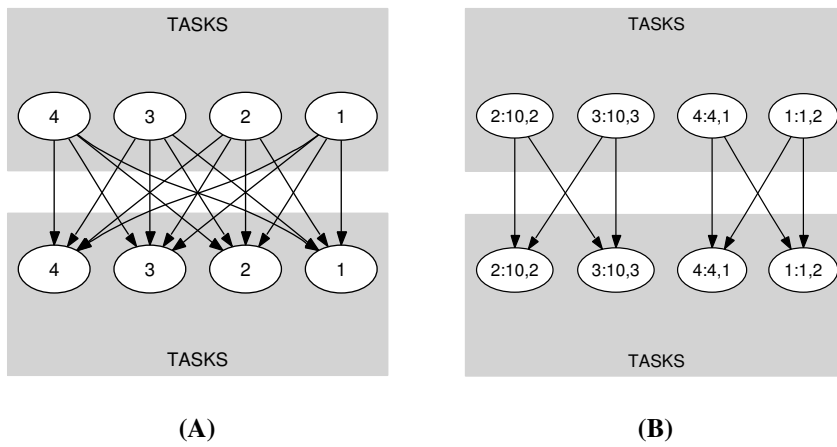


Figure 4.247: Initial and final graph of the interval_and_sum constraint

Graph model

We use a bipartite graph where each class of vertices corresponds to the different tasks of the TASKS collection. There is an arc between two tasks if their origins belong to the same interval. Finally we enforce a `sum_ctr` constraint on each set \mathcal{S} of successors of the different vertices of the final graph. This put a restriction on the maximum value of the sum of the `height` attributes of the tasks of \mathcal{S} .

Automaton

Figure 4.248 depicts the automaton associated to the `interval_and_sum` constraint. To each item of the collection TASKS corresponds a signature variable S_i , which is equal to 1.

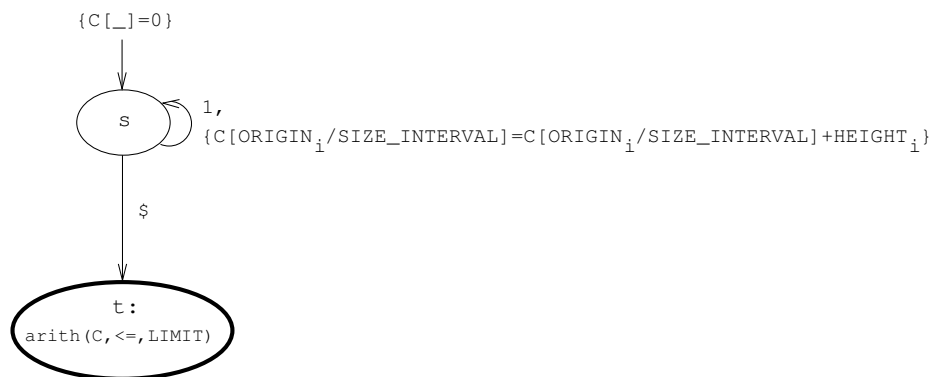


Figure 4.248: Automaton of the `interval_and_sum` constraint

Usage

This constraint can be use for timetabling problems. In this context the different intervals are interpreted as morning and afternoon periods of different consecutive days. We have a capacity constraint for all tasks that are assigned to the same morning or afternoon of a given day.

Key words

timetabling constraint, resource constraint, temporal constraint, assignment, interval, automaton, automaton with array of counters.

20000128

567

4.111 inverse

Origin	CHIP
Constraint	inverse(NODES)
Synonym(s)	assignment.
Argument(s)	NODES : collection(index – int, succ – dvar, pred – dvar)
Restriction(s)	<pre> required(NODES, [index, succ, pred]) NODES.index ≥ 1 NODES.index ≤ NODES distinct(NODES, index) NODES.succ ≥ 1 NODES.succ ≤ NODES NODES.pred ≥ 1 NODES.pred ≤ NODES </pre>
Purpose	<p>Enforce each vertex of a digraph to have exactly one predecessor and one successor. In addition the following property also holds: If the successor of the i^{th} node is the j^{th} node then the predecessor of the j^{th} node is the i^{th} node.</p>
<hr/>	
Arc input(s)	NODES
Arc generator	CLIQUE \mapsto collection(nodes1, nodes2)
Arc arity	2
Arc constraint(s)	<ul style="list-style-type: none"> • nodes1.succ = nodes2.index • nodes2.pred = nodes1.index
Graph property(ies)	NARC = NODES
<hr/>	
Example	$\text{inverse} \left(\left(\begin{array}{l} \text{index} - 1 \quad \text{succ} - 2 \quad \text{pred} - 2, \\ \text{index} - 2 \quad \text{succ} - 1 \quad \text{pred} - 1, \\ \text{index} - 3 \quad \text{succ} - 5 \quad \text{pred} - 4, \\ \text{index} - 4 \quad \text{succ} - 3 \quad \text{pred} - 5, \\ \text{index} - 5 \quad \text{succ} - 4 \quad \text{pred} - 3 \end{array} \right) \right)$
	<p>Parts (A) and (B) of Figure 4.249 respectively show the initial and final graph. Since we use the NARC graph property, the arcs of the final graph are stressed in bold.</p>
Graph model	<p>In order to express the binary constraint that links two vertices one has to make explicit the identifier of the vertices. This is why the inverse constraint considers objects that have three attributes:</p> <ul style="list-style-type: none"> • One fixed attribute index that is the identifier of the vertex, • One variable attribute succ that is the successor of the vertex,

- One variable attribute `pred` that is the predecessor of the vertex.

Signature

Since all the `index` attributes of the `NODES` collection are distinct and because of the first condition `nodes1.succ = nodes2.index` of the arc constraint all the vertices of the final graph have at most one predecessor.

Since all the `index` attributes of the `NODES` collection are distinct and because of the second condition `nodes2.pred = nodes1.index` of the arc constraint all the vertices of the final graph have at most one successor.

From the two previous remarks it follows that the final graph is made up from disjoint paths and disjoint circuits. Therefore the maximum number of arcs of the final graph is equal to its maximum number of vertices `NODES`. So we can rewrite the graph property $NARC = |NODES|$ to $NARC \geq |NODES|$ and simplify $NARC$ to \overline{NARC} .

Automaton

Figure 4.250 depicts the automaton associated to the `inverse` constraint. To each item of the collection `NODES` corresponds a signature variable S_i , which is equal to 1.

Usage

This constraint is used in order to make the link between the successor and the predecessor variables. This is sometimes required by specific heuristics that use both predecessor and successor variables. In some problems, the successor and predecessor variables are respectively interpreted as *column* and *row* variables. This is for instance the case in the *n*-queens problem (i.e. place *n* queens on a *n* by *n* chessboard in such a way that no two queens are on the same row, the same column or the same diagonal) when we use the following model: To each column of the chessboard we associate a variable which gives the row where the corresponding queen is located. Symmetrically, to each row of the chessboard we create a variable which indicates the column where the associated queen is placed. Having these two sets of variables, we can now write a heuristics which selects the column or the row for which we have the fewest number of alternatives for placing a queen.

Remark

In the original `inverse` constraint of CHIP the `index` attribute was not explicitly present. It was implicitly defined as the position of a variable in a list.

See also

`cycle`, `inverse_set`.

Key words

graph constraint, channeling constraint, permutation channel, permutation, dual model, *n*-queen, automaton, automaton with array of counters.

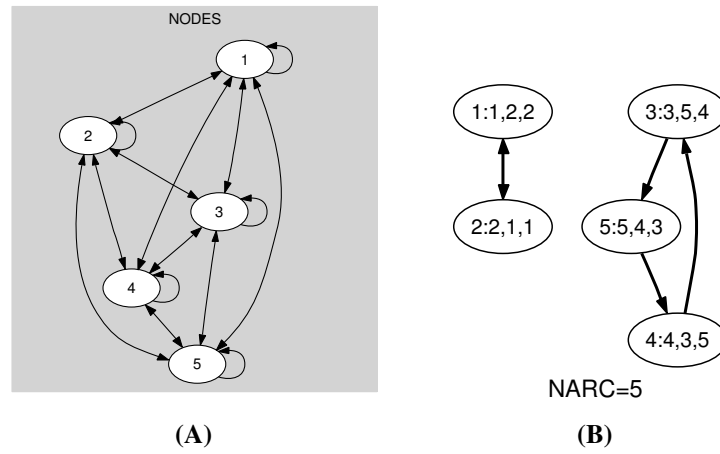


Figure 4.249: Initial and final graph of the inverse constraint

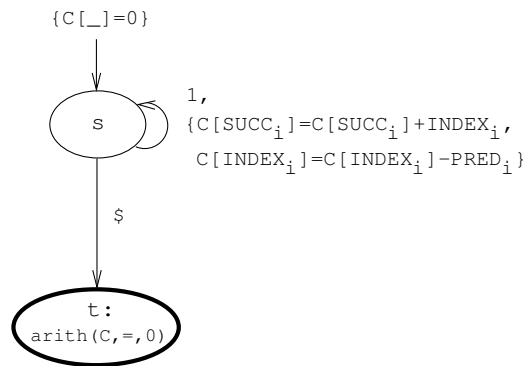


Figure 4.250: Automaton of the inverse constraint

20000128

571

4.112 inverse_set

Origin	Derived from <i>inverse</i> .
Constraint	<i>inverse_set</i> (X, Y)
Argument(s)	X : collection(index – int, set – svar) Y : collection(index – int, set – svar)
Restriction(s)	required(X, [index, set]) required(Y, [index, set]) increasing_seq(X, index) increasing_seq(Y, index) X.index ≥ 1 X.index ≤ Y Y.index ≥ 1 Y.index ≤ X X.set ≥ 1 X.set ≤ Y Y.set ≥ 1 Y.set ≤ X
Purpose	If value j belongs to the x set variable of the i^{th} item of the X collection then value i belongs also to the y set variable of the j^{th} item of the Y collection.
Arc input(s)	X Y
Arc generator	<i>PRODUCT</i> \mapsto collection(x, y)
Arc arity	2
Arc constraint(s)	in_set(y.index, x.set) \Leftrightarrow in_set(x.index, y.set)
Graph property(ies)	NARC = $ X * Y $

Example	$\text{inverse_set} \left(\left(\left\{ \begin{array}{ll} \text{index} - 1 & \text{set} - \{2, 4\}, \\ \text{index} - 2 & \text{set} - \{4\}, \\ \text{index} - 3 & \text{set} - \{1\}, \\ \text{index} - 4 & \text{set} - \{4\} \end{array} \right\}, \right. \right. \\ \left. \left. \left\{ \begin{array}{ll} \text{index} - 1 & \text{set} - \{3\}, \\ \text{index} - 2 & \text{set} - \{1\}, \\ \text{index} - 3 & \text{set} - \emptyset, \\ \text{index} - 4 & \text{set} - \{1, 2, 4\}, \\ \text{index} - 5 & \text{set} - \emptyset \end{array} \right\} \right) \right)$
----------------	--

Parts (A) and (B) of Figure 4.251 respectively show the initial and final graph. Since we use the **NARC** graph property, the arcs of the final graph are stressed in bold.

- Usage** The `inverse_set` constraint can for instance be used in order to model problems where one has to place items on a rectangular board in such a way that a column or a line can have more than one item. We have one set variable for each line of the board; Its values are the column indexes corresponding to the positions where an item is placed. Similarly we have also one set variable for each column of the board; Its values are the line indexes corresponding to the positions where an item is placed. The `inverse_set` constraint maintains the link between the lines and the columns variables. Figure 4.252 shows the board associated to the example.
- See also** `inverse`.
- Key words** channeling constraint, set channel, dual model, constraint involving set variables.

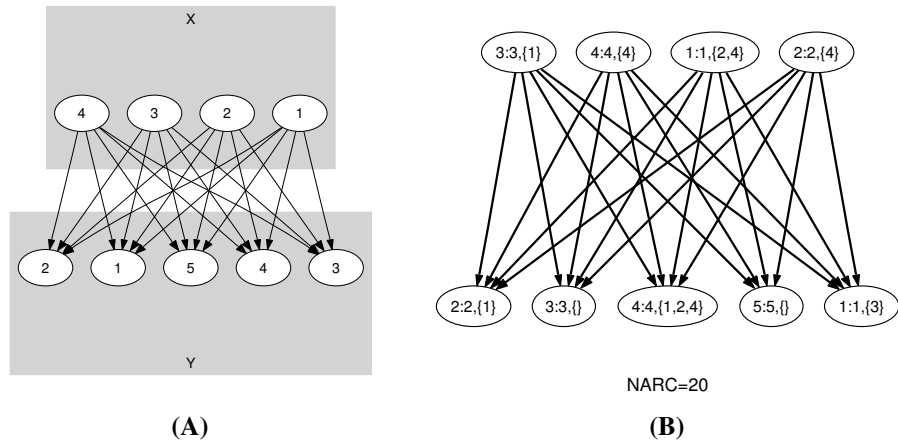


Figure 4.251: Initial and final graph of the `inverse_set` constraint

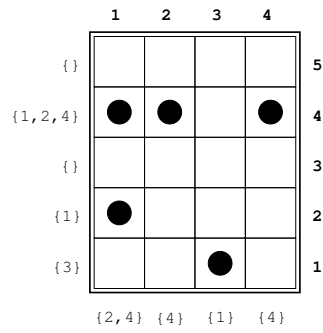


Figure 4.252: Board associated to the example

20041211

575

4.113 `ith_pos_different_from_0`

Origin	Used for defining the automaton of <code>min_n</code> .
Constraint	<code>ith_pos_different_from_0(ITH, POS, VARIABLES)</code>
Argument(s)	<pre> ITH : int POS : dvar VARIABLES : collection(var - dvar) </pre>
Restriction(s)	<pre> ITH ≥ 1 ITH ≤ VARIABLES POS ≥ ITH POS ≤ VARIABLES required(VARIABLES, var) </pre>

Purpose POS is the position of the ITH^{th} non-zero item of the sequence of variables VARIABLES.

Example `ith_pos_different_from_0` $\left(2, 4, \left\{ \begin{array}{l} \text{var} - 3, \\ \text{var} - 0, \\ \text{var} - 0, \\ \text{var} - 8, \\ \text{var} - 6 \end{array} \right\} \right)$

The previous constraint holds since 4 corresponds to the position of the 2^{th} non-zero item of the sequence 3 0 0 8 6.

Automaton Figure 4.253 depicts the automaton associated to the `ith_pos_different_from_0` constraint. To each variable VAR_i of the collection VARIABLES corresponds a 0-1 signature variable S_i . The following signature constraint links VAR_i and S_i : $VAR_i = 0 \Leftrightarrow S_i$.

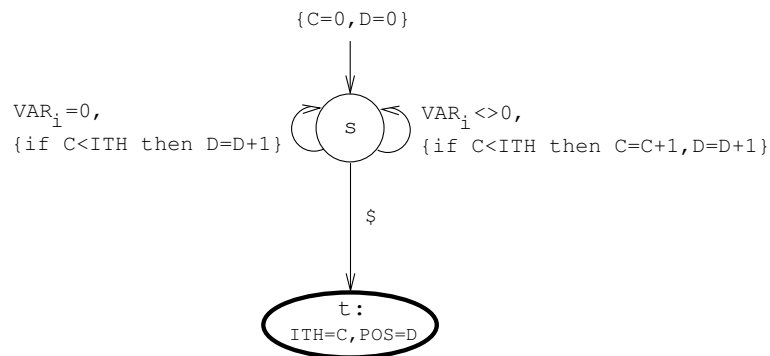


Figure 4.253: Automaton of the `ith_pos_different_from_0` constraint

See also `min_n`.

Key words

data constraint, table, joker value, automaton, automaton with counters, alpha-acyclic constraint network(3).

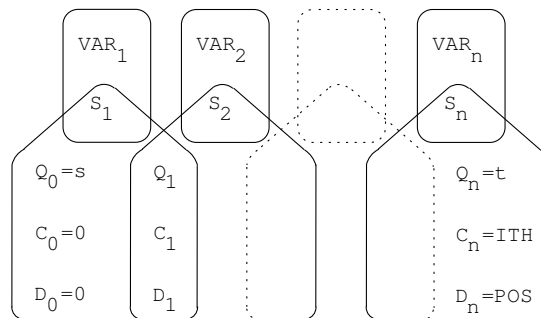
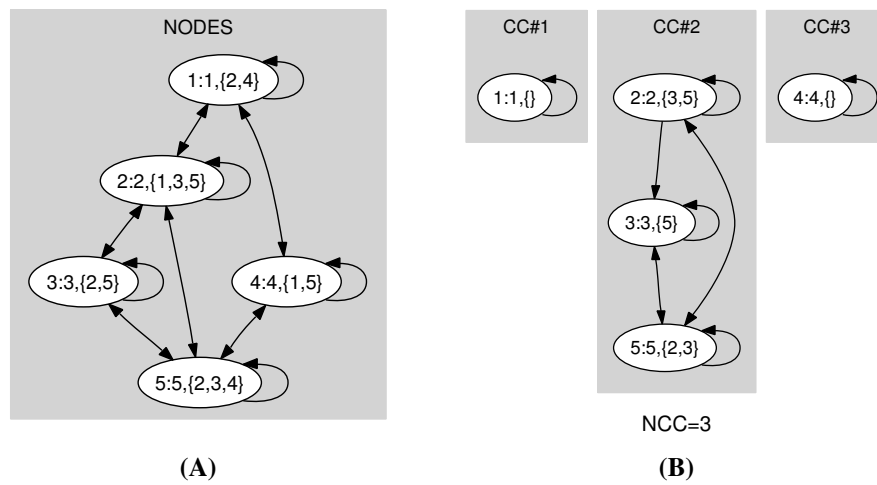


Figure 4.254: Hypergraph of the reformulation corresponding to the automaton of the `ith_pos_different_from_0` constraint

4.114 k_cut

Origin	E. Althaus
Constraint	<code>k_cut(K, NODES)</code>
Argument(s)	<code>K</code> : int <code>NODES</code> : <code>collection(index - int, succ - svar)</code>
Restriction(s)	$K \geq 1$ $K \leq \text{NODES} $ <code>required(NODES, [index, succ])</code> <code>NODES.index</code> ≥ 1 <code>NODES.index</code> $\leq \text{NODES} $ <code>distinct(NODES, index)</code>
Purpose	Select some arcs of a digraph in order to have at least K connected components (an isolated vertex is counted as one connected component).
Arc input(s)	<code>NODES</code>
Arc generator	<code>CLIQUE</code> \mapsto <code>collection(nodes1, nodes2)</code>
Arc arity	2
Arc constraint(s)	<code>nodes1.index = nodes2.index</code> \vee <code>in_set(nodes2.index, nodes1.succ)</code>
Graph property(ies)	$\text{NCC} \geq K$
Example	$\text{k_cut} \left(3, \left\{ \begin{array}{l} \text{index} - 1 \quad \text{succ} - \emptyset, \\ \text{index} - 2 \quad \text{succ} - \{3, 5\}, \\ \text{index} - 3 \quad \text{succ} - \{5\}, \\ \text{index} - 4 \quad \text{succ} - \emptyset, \\ \text{index} - 5 \quad \text{succ} - \{2, 3\} \end{array} \right\} \right)$ <p>Part (A) of Figure 4.255 shows the initial graph from which we have choose to start. It is derived from the set associated to each vertex. Each set describes the potential values of the <code>succ</code> attribute of a given vertex. Part (B) of Figure 4.255 gives the final graph associated to the example. The <code>k_cut</code> constraint holds since we have at least $K = 3$ connected components in the final graph.</p>
Graph model	<code>nodes1.index = nodes2.index</code> holds if <code>nodes1</code> and <code>nodes2</code> correspond to the same vertex. It is used in order to enforce keeping all the vertices of the initial graph. This is because an isolated vertex counts always as one connected component.
See also	<code>link_set_to_booleans</code> .
Key words	graph constraint, linear programming, connected component, constraint involving set variables.

Figure 4.255: Initial and final graph of the k_cut set constraint

4.115 lex2

Origin	[123]
Constraint	lex2(MATRIX)
Synonym(s)	double_lex, row_and_column_lex.
Type(s)	VECTOR : collection(var – dvar)
Argument(s)	MATRIX : collection(vec – VECTOR)
Restriction(s)	required(VECTOR, var) required(MATRIX, vec) same_size(MATRIX, vec)
Purpose	Given a matrix of domain variables, enforces that both adjacent rows, and adjacent columns are lexicographically ordered (adjacent rows and adjacent columns can be equal).
Example	$\text{lex2} \left(\left\{ \begin{array}{l} \text{vec} - \{\text{var} - 2, \text{var} - 2, \text{var} - 3\}, \\ \text{vec} - \{\text{var} - 2, \text{var} - 3, \text{var} - 1\} \end{array} \right\} \right)$
Usage	A <i>symmetry-breaking</i> constraint.
Remark	The idea of this <i>symmetry-breaking</i> constraint can already be found in the following articles of A.Lubiw [124, 125]. In block designs you sometimes want repeated blocks, so using the non-strict order would be required in this case.
See also	strict_lex2, allperm, lex_lesseq, lex_chain_lesseq.
Key words	predefined constraint, order constraint, matrix, matrix model, symmetry, matrix symmetry, lexicographic order.

4.116 `lex_alldifferent`

Origin	J. Pearson
Constraint	<code>lex_alldifferent(VECTORS)</code>
Synonym(s)	<code>lex_alldiff</code> , <code>lex_alldistinct</code> .
Type(s)	<code>VECTOR</code> : <code>collection(var – dvar)</code>
Argument(s)	<code>VECTORS</code> : <code>collection(vec – VECTOR)</code>
Restriction(s)	<code>required(VECTOR, var)</code> <code>required(VECTORS, vec)</code> <code>same_size(VECTORS, vec)</code>
Purpose	All the vectors of the collection <code>VECTORS</code> are distinct. Two vectors (u_1, u_2, \dots, u_n) and (v_1, v_2, \dots, v_n) are distinct if and only if there exist $i \in [1, n]$ such that $u_i \neq v_i$.
Arc input(s)	<code>VECTORS</code>
Arc generator	$\text{CLIQUE}(<) \mapsto \text{collection}(\text{vectors1}, \text{vectors2})$
Arc arity	2
Arc constraint(s)	<code>lex_different(vectors1.vec, vectors2.vec)</code>
Graph property(ies)	$\text{NARC} = \text{VECTORS} * (\text{VECTORS} - 1)/2$
Example	$\text{lex_alldifferent} \left(\left\{ \begin{array}{l} \text{vec} - \{\text{var} - 5, \text{var} - 2, \text{var} - 3\}, \\ \text{vec} - \{\text{var} - 5, \text{var} - 2, \text{var} - 6\}, \\ \text{vec} - \{\text{var} - 5, \text{var} - 3, \text{var} - 3\} \end{array} \right\} \right)$ <p>Parts (A) and (B) of Figure 4.256 respectively show the initial and final graph. Since we use the <code>NARC</code> graph property, the arcs of the final graph are stressed in bold.</p>
Signature	Since we use the <code>CLIQUE(<)</code> arc generator on the <code>VECTORS</code> collection the number of arcs of the initial graph is equal to $ \text{VECTORS} \cdot (\text{VECTORS} - 1)/2$. For this reason we can rewrite $\text{NARC} = \text{VECTORS} \cdot (\text{VECTORS} - 1)/2$ to $\text{NARC} \geq \text{VECTORS} \cdot (\text{VECTORS} - 1)/2$ and simplify $\overline{\text{NARC}}$ to NARC .
See also	<code>alldifferent</code> , <code>lex_different</code> .
Key words	decomposition, vector, bipartite matching.

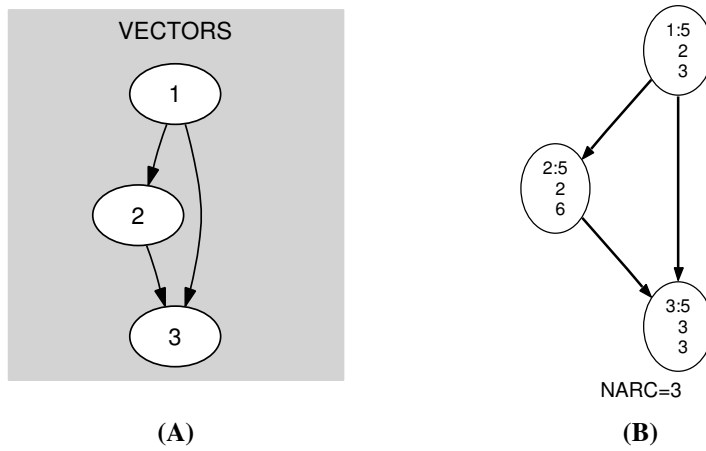


Figure 4.256: Initial and final graph of the `lex_alldifferent` constraint

4.117 lex_between

Origin	[126]
Constraint	<code>lex_between(LOWER_BOUND, VECTOR, UPPER_BOUND)</code>
Argument(s)	<code>LOWER_BOUND</code> : <code>collection(var - int)</code> <code>VECTOR</code> : <code>collection(var - dvar)</code> <code>UPPER_BOUND</code> : <code>collection(var - int)</code>
Restriction(s)	<code>required(LOWER_BOUND, var)</code> <code>required(VECTOR, var)</code> <code>required(UPPER_BOUND, var)</code> <code> LOWER_BOUND = VECTOR </code> <code> UPPER_BOUND = VECTOR </code> <code>lex_lesseq(LOWER_BOUND, VECTOR)</code> <code>lex_lesseq(VECTOR, UPPER_BOUND)</code>
Purpose	The vector VECTOR is lexicographically greater than or equal to the fixed vector LOWER_BOUND and lexicographically smaller than or equal to the fixed vector UPPER_BOUND.
Example	$\text{lex_between} \left(\begin{array}{l} \{\text{var} - 5, \text{var} - 2, \text{var} - 3, \text{var} - 9\}, \\ \{\text{var} - 5, \text{var} - 2, \text{var} - 6, \text{var} - 2\}, \\ \{\text{var} - 5, \text{var} - 2, \text{var} - 6, \text{var} - 3\} \end{array} \right)$
Automaton	<p>Figure 4.257 depicts the automaton associated to the <code>lex_between</code> constraint. Let L_i, V_i and U_i respectively be the <code>var</code> attributes of the i^{th} items of the <code>LOWER_BOUND</code>, the <code>VECTOR</code> and the <code>UPPER_BOUND</code> collections. To each triple (L_i, V_i, U_i) corresponds a signature variable S_i as well as the following signature constraint:</p> $\begin{aligned} (L_i < V_i) \wedge (V_i < U_i) &\Leftrightarrow S_i = 0 \wedge \\ (L_i < V_i) \wedge (V_i = U_i) &\Leftrightarrow S_i = 1 \wedge \\ (L_i < V_i) \wedge (V_i > U_i) &\Leftrightarrow S_i = 2 \wedge \\ (L_i = V_i) \wedge (V_i < U_i) &\Leftrightarrow S_i = 3 \wedge \\ (L_i = V_i) \wedge (V_i = U_i) &\Leftrightarrow S_i = 4 \wedge \\ (L_i = V_i) \wedge (V_i > U_i) &\Leftrightarrow S_i = 5 \wedge \\ (L_i > V_i) \wedge (V_i < U_i) &\Leftrightarrow S_i = 6 \wedge \\ (L_i > V_i) \wedge (V_i = U_i) &\Leftrightarrow S_i = 7 \wedge \\ (L_i > V_i) \wedge (V_i > U_i) &\Leftrightarrow S_i = 8. \end{aligned}$
Usage	<p>This constraint does usually not occur explicitly in practice. However it shows up indirectly in the context of the <code>lex_chain_less</code> and the <code>lex_chain_lesseq</code> constraints: In order to have a complete filtering algorithm for the <code>lex_chain_less</code> and the <code>lex_chain_lesseq</code> constraints one has to come up with a complete filtering algorithm for the <code>lex_between</code></p>

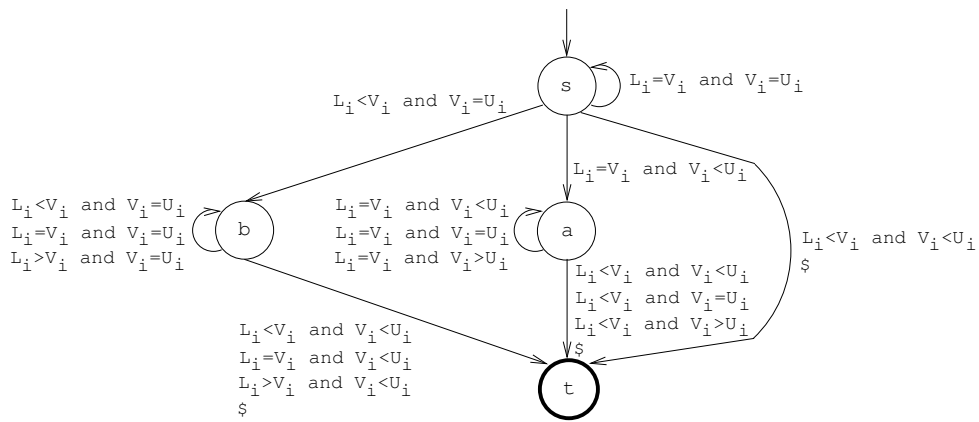


Figure 4.257: Automaton of the `lex_between` constraint

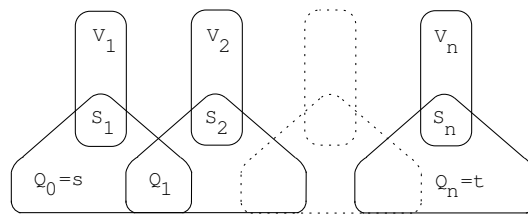


Figure 4.258: Hypergraph of the reformulation corresponding to the automaton of the `lex_between` constraint

constraint. The reason is that the `lex_chain_less` as well as the `lex_chain_lesseq` constraints both compute feasible lower and upper bounds for each vector they mention. Therefore one ends up with a `lex_between` constraint for each vector of the `lex_chain_less` and `lex_chain_lesseq` constraints.

Algorithm

[126].

See also

`lex_less`, `lex_lesseq`, `lex_greater`, `lex_greatereq`, `lex_chain_less`, `lex_chain_lesseq`.

Key words

order constraint, vector, symmetry, lexicographic order, Berge-acyclic constraint network, automaton, automaton without counters.

20030820

587

4.118 `lex_chain_less`

Origin	[126]
Constraint	<code>lex_chain_less(VECTORS)</code>
Usual name	<code>lex_chain</code>
Type(s)	<code>VECTOR : collection(var - dvar)</code>
Argument(s)	<code>VECTORS : collection(vec - VECTOR)</code>
Restriction(s)	<code>required(VECTOR, var)</code> <code>required(VECTORS, vec)</code> <code>same_size(VECTORS, vec)</code>

Purpose

For each pair of consecutive vectors VECTOR_i and VECTOR_{i+1} of the VECTORS collection we have that VECTOR_i is lexicographically strictly less than VECTOR_{i+1} . Given two vectors, \vec{X} and \vec{Y} of n components, $\langle X_0, \dots, X_n \rangle$ and $\langle Y_0, \dots, Y_n \rangle$, \vec{X} is *lexicographically strictly less than* \vec{Y} if and only if $X_0 < Y_0$ or $X_0 = Y_0$ and $\langle X_1, \dots, X_n \rangle$ is lexicographically strictly less than $\langle Y_1, \dots, Y_n \rangle$.

Arc input(s)	<code>VECTORS</code>
Arc generator	$\text{PATH} \mapsto \text{collection}(\text{vectors1}, \text{vectors2})$
Arc arity	2
Arc constraint(s)	<code>lex_less(vectors1.vec, vectors2.vec)</code>
Graph property(ies)	$\text{NARC} = \text{VECTORS} - 1$

Example	$\text{lex_chain_less} \left(\left(\left(\text{vec} - \begin{Bmatrix} \text{var} - 5, \\ \text{var} - 2, \\ \text{var} - 3, \\ \text{var} - 9 \end{Bmatrix}, \right) \right) \right)$
----------------	--

Parts (A) and (B) of Figure 4.259 respectively show the initial and final graph. Since we use the **NARC** graph property, the arcs of the final graph are stressed in bold. The `lex_chain_less` constraint holds since all the arc constraints of the initial graph are satisfied.

Signature	Since we use the <i>PATH</i> arc generator on the <i>VECTORS</i> collection the number of arcs of the initial graph is equal to $ \text{VECTORS} - 1$. For this reason we can rewrite $\text{NARC} = \text{VECTORS} - 1$ to $\text{NARC} \geq \text{VECTORS} - 1$ and simplify $\overline{\text{NARC}}$ to NARC .
Usage	This constraint was motivated for breaking symmetry: More precisely when one wants to lexicographically order the consecutive columns of a matrix of decision variables. A further motivation is that using a set of lexicographic ordering constraints between two vectors does usually not allow to come up with a complete pruning.
Algorithm	A complete filtering algorithm for a chain of lexicographical constraints is presented in [126].
See also	<code>lex_between</code> , <code>lex_chain_lesseq</code> , <code>lex_less</code> , <code>lex_lesseq</code> , <code>lex_greater</code> , <code>lex_greatereq</code> .
Key words	decomposition, order constraint, vector, symmetry, matrix symmetry, lexicographic order.

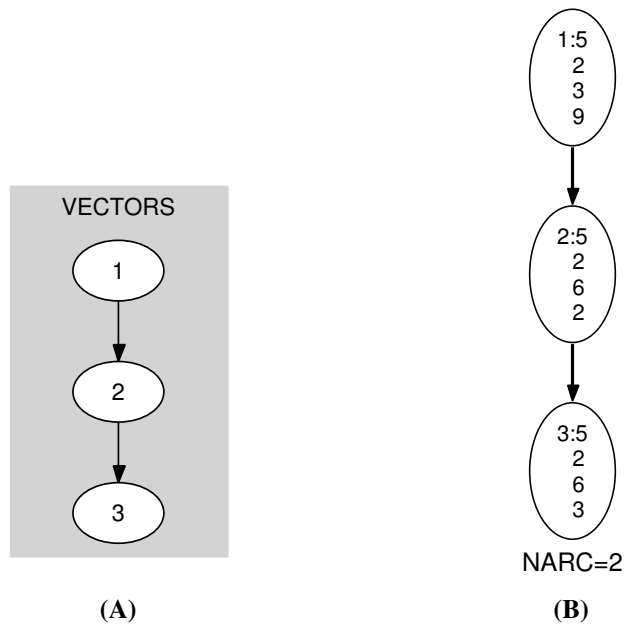


Figure 4.259: Initial and final graph of the `lex_chain_less` constraint

20030820

591

4.119 `lex_chain_lesseq`

Origin	[126]
Constraint	<code>lex_chain_lesseq(VECTORS)</code>
Usual name	<code>lex_chain</code>
Type(s)	<code>VECTOR : collection(var - dvar)</code>
Argument(s)	<code>VECTORS : collection(vec - VECTOR)</code>
Restriction(s)	<code>required(VECTOR, var)</code> <code>required(VECTORS, vec)</code> <code>same_size(VECTORS, vec)</code>

Purpose

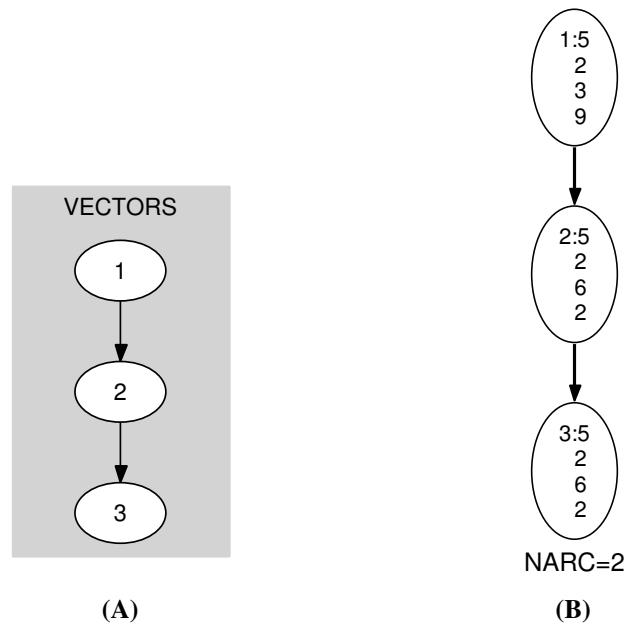
For each pair of consecutive vectors VECTOR_i and VECTOR_{i+1} of the `VECTORS` collection we have that VECTOR_i is lexicographically less than or equal to VECTOR_{i+1} . Given two vectors, \vec{X} and \vec{Y} of n components, $\langle X_0, \dots, X_n \rangle$ and $\langle Y_0, \dots, Y_n \rangle$, \vec{X} is *lexicographically less than or equal to* \vec{Y} if and only if $n = 0$ or $X_0 < Y_0$ or $X_0 = Y_0$ and $\langle X_1, \dots, X_n \rangle$ is lexicographically less than or equal to $\langle Y_1, \dots, Y_n \rangle$.

Arc input(s)	<code>VECTORS</code>
Arc generator	<code>PATH</code> \mapsto <code>collection(vectors1, vectors2)</code>
Arc arity	2
Arc constraint(s)	<code>lex_lesseq(vectors1.vec, vectors2.vec)</code>
Graph property(ies)	<code>NARC = VECTORS - 1</code>

Example	$\text{lex_chain_lesseq} \left(\left(\left(\text{vec} - \begin{Bmatrix} \text{var} - 5, \\ \text{var} - 2, \\ \text{var} - 3, \\ \text{var} - 9 \end{Bmatrix} \right), \right. \right. \\ \left. \left. \left(\text{vec} - \begin{Bmatrix} \text{var} - 5, \\ \text{var} - 2, \\ \text{var} - 6, \\ \text{var} - 2 \end{Bmatrix} \right), \right. \right. \\ \left. \left. \left(\text{vec} - \begin{Bmatrix} \text{var} - 5, \\ \text{var} - 2, \\ \text{var} - 6, \\ \text{var} - 2 \end{Bmatrix} \right) \right) \right)$
----------------	---

Parts (A) and (B) of Figure 4.260 respectively show the initial and final graph. Since we use the **NARC** graph property, the arcs of the final graph are stressed in bold. The `lex_chain_lesseq` constraint holds since all the arc constraints of the initial graph are satisfied.

Signature	Since we use the <i>PATH</i> arc generator on the <i>VECTORS</i> collection the number of arcs of the initial graph is equal to $ \text{VECTORS} - 1$. For this reason we can rewrite $\text{NARC} = \text{VECTORS} - 1$ to $\text{NARC} \geq \text{VECTORS} - 1$ and simplify $\overline{\text{NARC}}$ to NARC .
Usage	This constraint was motivated for breaking symmetry: More precisely when one wants to lexicographically order the consecutive columns of a matrix of decision variables. A further motivation is that using a set of lexicographic ordering constraints between two vectors does usually not allow to come up with a complete pruning.
Algorithm	A complete filtering algorithm for a chain of lexicographical constraints is presented in [126].
See also	<code>lex_between</code> , <code>lex_chain_less</code> , <code>lex_less</code> , <code>lex_lesseq</code> , <code>lex_greater</code> , <code>lex_greatereq</code> .
Key words	decomposition, order constraint, vector, symmetry, matrix symmetry, lexicographic order.

Figure 4.260: Initial and final graph of the `lex_chain_lesseq` constraint

20030820

595

4.120 `lex_different`

Origin	Used for defining <code>lex_alldifferent</code> .
Constraint	<code>lex_different(VECTOR1, VECTOR2)</code>
Argument(s)	<p><code>VECTOR1</code> : <code>collection(var - dvar)</code> <code>VECTOR2</code> : <code>collection(var - dvar)</code></p>
Restriction(s)	<p><code>required(VECTOR1, var)</code> <code>required(VECTOR2, var)</code> $\text{VECTOR1} = \text{VECTOR2}$</p>
Purpose	<p>Vectors <code>VECTOR1</code> and <code>VECTOR2</code> differ from at least one component.</p>
<hr/>	
Arc input(s)	<code>VECTOR1 VECTOR2</code>
Arc generator	$\text{PRODUCT}(=) \mapsto \text{collection}(\text{vector1}, \text{vector2})$
Arc arity	2
Arc constraint(s)	<code>vector1.var \neq vector2.var</code>
Graph property(ies)	$\text{NARC} \geq 1$
<hr/>	
Example	<p><code>lex_different</code> $\left(\begin{array}{l} \{\text{var} - 5, \text{var} - 2, \text{var} - 7, \text{var} - 1\}, \\ \{\text{var} - 5, \text{var} - 3, \text{var} - 7, \text{var} - 1\} \end{array} \right)$</p> <p>Parts (A) and (B) of Figure 4.261 respectively show the initial and final graph. Since we use the <code>NARC</code> graph property, the unique arc of the final graph is stressed in bold. It corresponds to a component where the two vectors differ.</p>
Automaton	Figure 4.262 depicts the automaton associated to the <code>lex_different</code> constraint. Let VAR1_i and VAR2_i respectively be the <code>var</code> attributes of the i^{th} items of the <code>VECTOR1</code> and the <code>VECTOR2</code> collections. To each pair $(\text{VAR1}_i, \text{VAR2}_i)$ corresponds a 0-1 signature variable S_i as well as the following signature constraint: $\text{VAR1}_i = \text{VAR2}_i \Leftrightarrow S_i$.
Used in	<code>lex_alldifferent</code> .
See also	<code>lex_greatereq</code> , <code>lex_less</code> , <code>lex_lesseq</code> .
Key words	vector, disequality, Berge-acyclic constraint network, automaton, automaton without counters.

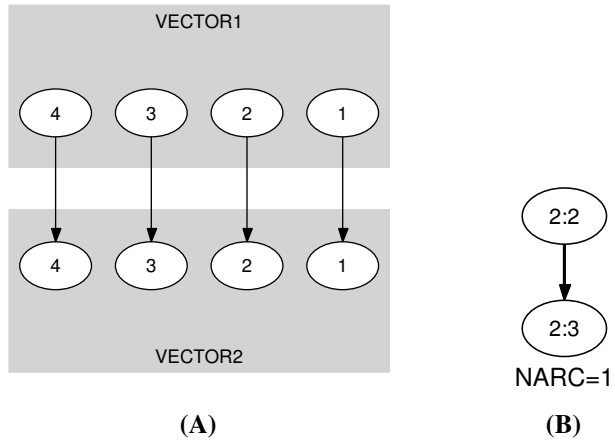


Figure 4.261: Initial and final graph of the `lex_different` constraint

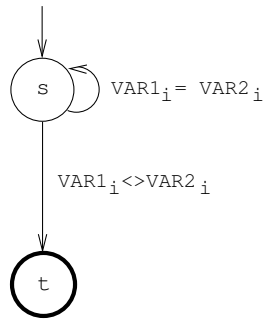


Figure 4.262: Automaton of the `lex_different` constraint

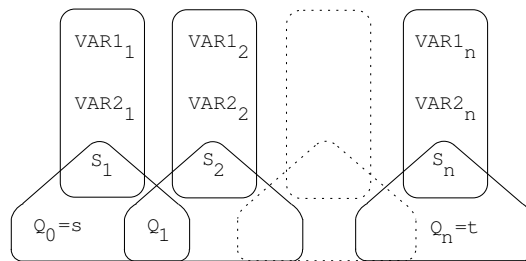


Figure 4.263: Hypergraph of the reformulation corresponding to the automaton of the `lex_different` constraint

4.121 lex_greater

Origin	CHIP
Constraint	$lex_greater(VECTOR1, VECTOR2)$
Argument(s)	$VECTOR1 : collection(var - dvar)$ $VECTOR2 : collection(var - dvar)$
Restriction(s)	$required(VECTOR1, var)$ $required(VECTOR2, var)$ $ VECTOR1 = VECTOR2 $
Purpose	$VECTOR1$ is lexicographically strictly greater than $VECTOR2$. Given two vectors, \vec{X} and \vec{Y} of n components, $\langle X_0, \dots, X_n \rangle$ and $\langle Y_0, \dots, Y_n \rangle$, \vec{X} is <i>lexicographically strictly greater than</i> \vec{Y} if and only if $X_0 > Y_0$ or $X_0 = Y_0$ and $\langle X_1, \dots, X_n \rangle$ is lexicographically strictly greater than $\langle Y_1, \dots, Y_n \rangle$.
Derived Collection(s)	$col \left(\begin{array}{l} DESTINATION - collection(index - int, x - int, y - int), \\ [item(index - 0, x - 0, y - 0)] \end{array} \right)$ $col \left(\begin{array}{l} COMPONENTS - collection(index - int, x - dvar, y - dvar), \\ [item(index - VECTOR1.key, x - VECTOR1.var, y - VECTOR2.var)] \end{array} \right)$
Arc input(s)	COMPONENTS DESTINATION
Arc generator	$PRODUCT(PATH, VOID) \mapsto collection(item1, item2)$
Arc arity	2
Arc constraint(s)	$item2.index > 0 \wedge item1.x = item1.y \vee item2.index = 0 \wedge item1.x > item1.y$
Graph property(ies)	$PATH.FROM.TO(index, 1, 0) = 1$

Example $lex_greater \left(\begin{array}{l} \{\mathbf{var - 5}, \mathbf{var - 2}, \mathbf{var - 7}, \mathbf{var - 1}\}, \\ \{\mathbf{var - 5}, \mathbf{var - 2}, \mathbf{var - 6}, \mathbf{var - 2}\} \end{array} \right)$

Parts (A) and (B) of Figure 4.264 respectively show the initial and final graph. Since we use the **PATH.FROM.TO** graph property we show the following information on the final graph:

- The vertices which respectively correspond to the start and the end of the required path are stressed in bold.
- The arcs on the required path are also stressed in bold.

Graph model The vertices of the initial graph are generated in the following way:

- We create a vertex c_i for each pair of components which both have the same index i .
- We create an additional dummy vertex called d .

The arcs of the initial graph are generated in the following way:

- We create an arc between c_i and d . We associate to this arc the arc constraint $\text{item}_1.x > \text{item}_2.y$.
- We create an arc between c_i and c_{i+1} . We associate to this arc the arc constraint $\text{item}_1.x = \text{item}_2.y$.

The `lex_greater` constraint holds when there exist a path from c_1 to d . This path can be interpreted as a sequence of *equality* constraints on the prefix of both vectors, immediately followed by a *greater than* constraint.

Signature	Since the maximum value returned by the graph property <code>PATH_FROM_TO</code> is equal to 1 we can rewrite <code>PATH_FROM_TO(index, 1, 0) = 1</code> to <code>PATH_FROM_TO(index, 1, 0) ≥ 1</code> . Therefore we simplify <u><code>PATH_FROM_TO</code></u> to <code>PATH_FROM_TO</code> .
Automaton	Figure 4.265 depicts the automaton associated to the <code>lex_greater</code> constraint. Let VAR1_i and VAR2_i respectively be the <code>var</code> attributes of the i^{th} items of the <code>VECTOR1</code> and the <code>VECTOR2</code> collections. To each pair $(\text{VAR1}_i, \text{VAR2}_i)$ corresponds a signature variable S_i as well as the following signature constraint: $(\text{VAR1}_i < \text{VAR2}_i \Leftrightarrow S_i = 1) \wedge (\text{VAR1}_i = \text{VAR2}_i \Leftrightarrow S_i = 2) \wedge (\text{VAR1}_i > \text{VAR2}_i \Leftrightarrow S_i = 3)$.
Remark	A <i>multiset ordering</i> constraint and its corresponding filtering algorithm are described in [127].
Algorithm	The first complete filtering algorithm for this constraint was presented in [36]. A second complete filtering algorithm, detecting entailment in a more eager way, was given in [128]. This second algorithm was derived from a deterministic finite automata. A third complete filtering algorithm extending the algorithm presented in [36] detecting entailment is given in the PhD thesis of Z.Kızıltan [129, page 95]. The previous thesis [129, pages 105–109] presents also a filtering algorithm handling the fact that a given variable has more than one occurrence.
See also	<code>lex_between</code> , <code>lex_greatereq</code> , <code>lex_less</code> , <code>lex_lesseq</code> , <code>lex_chain_less</code> , <code>lex_chain_lesseq</code> .
Key words	order constraint, vector, symmetry, matrix symmetry, lexicographic order, multiset ordering, duplicated variables, Berge-acyclic constraint network, automaton, automaton without counters, derived collection.

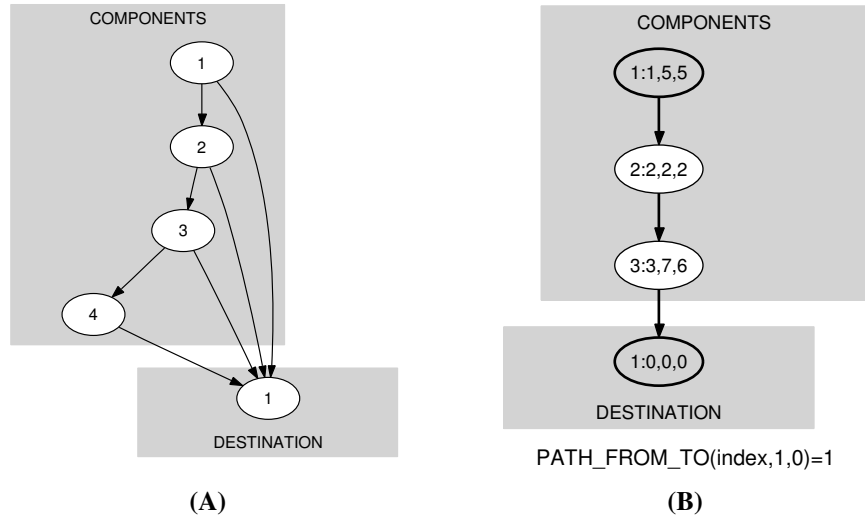


Figure 4.264: Initial and final graph of the `lex_greater` constraint

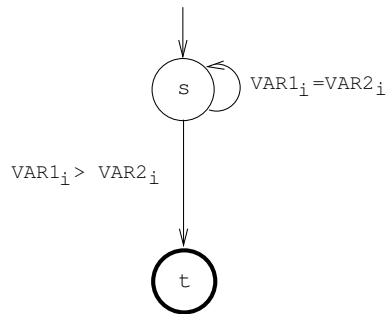


Figure 4.265: Automaton of the `lex_greater` constraint

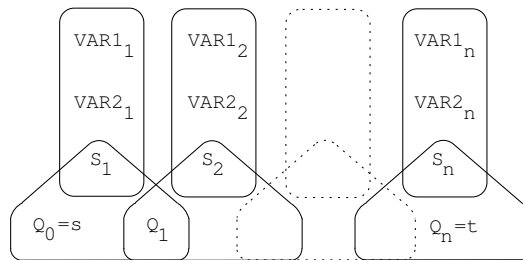


Figure 4.266: Hypergraph of the reformulation corresponding to the automaton of the `lex_greater` constraint

20030820

601

4.122 lex_greatereq

Origin	CHIP
Constraint	<code>lex_greatereq(VECTOR1, VECTOR2)</code>
Argument(s)	VECTOR1 : collection(var – dvar) VECTOR2 : collection(var – dvar)
Restriction(s)	required(VECTOR1, var) required(VECTOR2, var) $ \text{VECTOR1} = \text{VECTOR2} $
Purpose	<p>VECTOR1 is lexicographically greater than or equal to VECTOR2. Given two vectors, \vec{X} and \vec{Y} of n components, $\langle X_0, \dots, X_n \rangle$ and $\langle Y_0, \dots, Y_n \rangle$, \vec{X} is <i>lexicographically greater than or equal to</i> \vec{Y} if and only if $n = 0$ or $X_0 > Y_0$ or $X_0 = Y_0$ and $\langle X_1, \dots, X_n \rangle$ is lexicographically greater than or equal to $\langle Y_1, \dots, Y_n \rangle$.</p>
Derived Collection(s)	$\text{col} \left(\begin{array}{l} \text{DESTINATION} - \text{collection}(\text{index} - \text{int}, \text{x} - \text{int}, \text{y} - \text{int}), \\ [\text{item}(\text{index} - 0, \text{x} - 0, \text{y} - 0)] \end{array} \right)$ $\text{col} \left(\begin{array}{l} \text{COMPONENTS} - \text{collection}(\text{index} - \text{int}, \text{x} - \text{dvar}, \text{y} - \text{dvar}), \\ [\text{item}(\text{index} - \text{VECTOR1.key}, \text{x} - \text{VECTOR1.var}, \text{y} - \text{VECTOR2.var})] \end{array} \right)$
Arc input(s)	COMPONENTS DESTINATION
Arc generator	$\text{PRODUCT}(\text{PATH}, \text{VOID}) \mapsto \text{collection}(\text{item1}, \text{item2})$
Arc arity	2
Arc constraint(s)	$\bigvee \left(\begin{array}{l} \text{item2.index} > 0 \wedge \text{item1.x} = \text{item1.y}, \\ \text{item1.index} < \text{VECTOR1} \wedge \text{item2.index} = 0 \wedge \text{item1.x} > \text{item1.y}, \\ \text{item1.index} = \text{VECTOR1} \wedge \text{item2.index} = 0 \wedge \text{item1.x} \geq \text{item1.y} \end{array} \right)$
Graph property(ies)	$\text{PATH_FROM_TO}(\text{index}, 1, 0) = 1$
Example	$\text{lex_greatereq} \left(\begin{array}{l} \{\text{var} - 5, \text{var} - 2, \text{var} - 8, \text{var} - 9\}, \\ \{\text{var} - 5, \text{var} - 2, \text{var} - 6, \text{var} - 2\} \end{array} \right)$ $\text{lex_greatereq} \left(\begin{array}{l} \{\text{var} - 5, \text{var} - 2, \text{var} - 3, \text{var} - 9\}, \\ \{\text{var} - 5, \text{var} - 2, \text{var} - 3, \text{var} - 9\} \end{array} \right)$

Parts (A) and (B) of Figure 4.267 respectively show the initial and final graph associated to the first example. Since we use the **PATH_FROM_TO** graph property we show on the final graph the following information:

- The vertices which respectively correspond to the start and the end of the required path are stressed in bold.
- The arcs on the required path are also stressed in bold.

Graph model

The vertices of the initial graph are generated in the following way:

- We create a vertex c_i for each pair of components which both have the same index i .
- We create an additional dummy vertex called d .

The arcs of the initial graph are generated in the following way:

- We create an arc between c_i and d . When c_i was generated from the last components of both vectors We associate to this arc the arc constraint $\text{item}_1.x \geq \text{item}_2.y$; Otherwise we associate to this arc the arc constraint $\text{item}_1.x > \text{item}_2.y$;
- We create an arc between c_i and c_{i+1} . We associate to this arc the arc constraint $\text{item}_1.x = \text{item}_2.y$.

The `lex_greatereq` constraint holds when there exist a path from c_1 to d . This path can be interpreted as a maximum sequence of *equality* constraints on the prefix of both vectors, eventually followed by a *greater than* constraint.

Signature

Since the maximum value returned by the graph property `PATH_FROM_TO` is equal to 1 we can rewrite `PATH_FROM_TO(index, 1, 0) = 1` to `PATH_FROM_TO(index, 1, 0) ≥ 1`. Therefore we simplify `PATH_FROM_TO` to `PATH_FROM_TO`.

Automaton

Figure 4.268 depicts the automaton associated to the `lex_greatereq` constraint. Let VAR1_i and VAR2_i respectively be the `var` attributes of the i^{th} items of the `VECTOR1` and the `VECTOR2` collections. To each pair $(\text{VAR1}_i, \text{VAR2}_i)$ corresponds a signature variable S_i as well as the following signature constraint: $(\text{VAR1}_i < \text{VAR2}_i \Leftrightarrow S_i = 1) \wedge (\text{VAR1}_i = \text{VAR2}_i \Leftrightarrow S_i = 2) \wedge (\text{VAR1}_i > \text{VAR2}_i \Leftrightarrow S_i = 3)$.

Remark

A *multiset ordering* constraint and its corresponding filtering algorithm are described in [127].

Algorithm

The first complete filtering algorithm for this constraint was presented in [36]. A second complete filtering algorithm, detecting entailment in a more eager way, was given in [128]. This second algorithm was derived from a deterministic finite automata. A third complete filtering algorithm extending the algorithm presented in [36] detecting entailment is given in the PhD thesis of Z.Kızıltan [129, page 95]. The previous thesis [129, pages 105–109] presents also a filtering algorithm handling the fact that a given variable has more than one occurrence.

See also

`lex_between`, `lex_greater`, `lex_less`, `lex_lesseq`, `lex_chain_less`, `lex_chain_lesseq`.

Key words

order constraint, vector, symmetry, matrix symmetry, lexicographic order, multiset ordering, duplicated variables, Berge-acyclic constraint network, automaton, automaton without counters, derived collection.

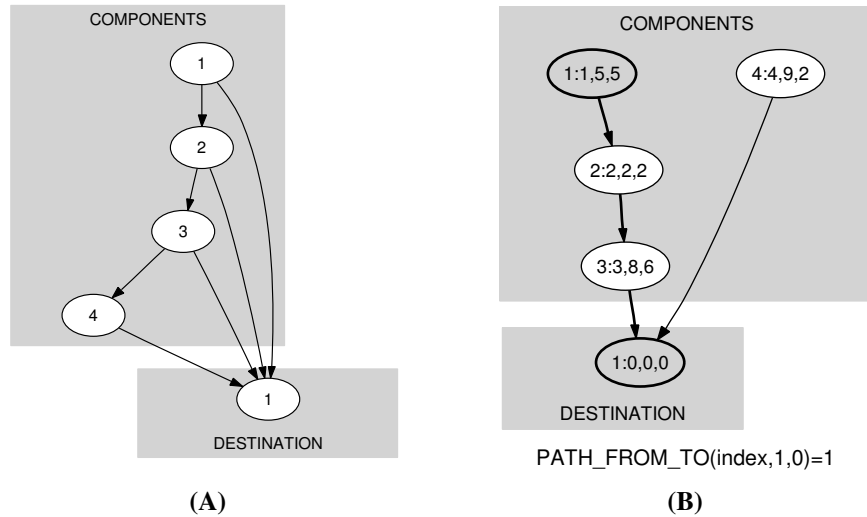


Figure 4.267: Initial and final graph of the *lex_greatereq* constraint

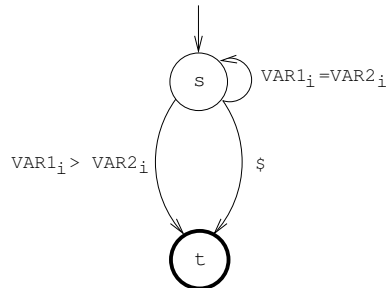


Figure 4.268: Automaton of the *lex_greatereq* constraint

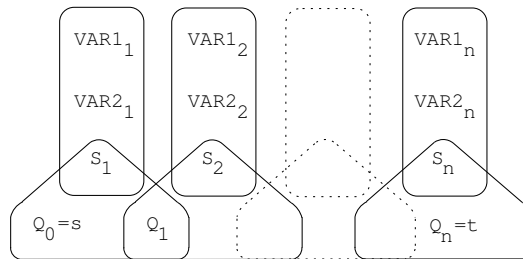


Figure 4.269: Hypergraph of the reformulation corresponding to the automaton of the *lex_greatereq* constraint

20030820

605

4.123 **lex_less**

Origin	CHIP
Constraint	<code>lex_less(VECTOR1, VECTOR2)</code>
Argument(s)	<p>VECTOR1 : <code>collection(var - dvar)</code> VECTOR2 : <code>collection(var - dvar)</code></p>
Restriction(s)	<p><code>required(VECTOR1, var)</code> <code>required(VECTOR2, var)</code> <code> VECTOR1 = VECTOR2 </code></p>
Purpose	<p>VECTOR1 is lexicographically strictly less than VECTOR2. Given two vectors, \vec{X} and \vec{Y} of n components, $\langle X_0, \dots, X_n \rangle$ and $\langle Y_0, \dots, Y_n \rangle$, \vec{X} is <i>lexicographically strictly less than</i> \vec{Y} if and only if $X_0 < Y_0$ or $X_0 = Y_0$ and $\langle X_1, \dots, X_n \rangle$ is lexicographically strictly less than $\langle Y_1, \dots, Y_n \rangle$.</p>
Derived Collection(s)	$\text{col} \left(\begin{array}{l} \text{DESTINATION} - \text{collection}(\text{index} - \text{int}, \text{x} - \text{int}, \text{y} - \text{int}), \\ [\text{item}(\text{index} - 0, \text{x} - 0, \text{y} - 0)] \end{array} \right)$ <hr/> $\text{col} \left(\begin{array}{l} \text{COMPONENTS} - \text{collection}(\text{index} - \text{int}, \text{x} - \text{dvar}, \text{y} - \text{dvar}), \\ [\text{item}(\text{index} - \text{VECTOR1.key}, \text{x} - \text{VECTOR1.var}, \text{y} - \text{VECTOR2.var})] \end{array} \right)$
Arc input(s)	COMPONENTS DESTINATION
Arc generator	$\text{PRODUCT}(\text{PATH}, \text{VOID}) \mapsto \text{collection}(\text{item1}, \text{item2})$
Arc arity	2
Arc constraint(s)	$\text{item2.index} > 0 \wedge \text{item1.x} = \text{item1.y} \vee \text{item2.index} = 0 \wedge \text{item1.x} < \text{item1.y}$
Graph property(ies)	$\text{PATH_FROM_TO}(\text{index}, 1, 0) = 1$
Example	$\text{lex_less} \left(\begin{array}{l} \{\text{var} - 5, \text{var} - 2, \text{var} - 3, \text{var} - 9\}, \\ \{\text{var} - 5, \text{var} - 2, \text{var} - 6, \text{var} - 2\} \end{array} \right)$ <p>Parts (A) and (B) of Figure 4.270 respectively show the initial and final graph. Since we use the PATH_FROM_TO graph property we show on the final graph the following information:</p> <ul style="list-style-type: none"> • The vertices which respectively correspond to the start and the end of the required path are stressed in bold. • The arcs on the required path are also stressed in bold.
Graph model	<p>The vertices of the initial graph are generated in the following way:</p> <ul style="list-style-type: none"> • We create a vertex c_i for each pair of components which both have the same index i. • We create an additional dummy vertex called d.

The arcs of the initial graph are generated in the following way:

- We create an arc between c_i and d . We associate to this arc the arc constraint $\text{item}_1.x < \text{item}_2.y$.
- We create an arc between c_i and c_{i+1} . We associate to this arc the arc constraint $\text{item}_1.x = \text{item}_2.y$.

The `lex_less` constraint holds when there exist a path from c_1 to d . This path can be interpreted as a sequence of *equality* constraints on the prefix of both vectors, immediately followed by a *less than* constraint.

Signature	Since the maximum value returned by the graph property <code>PATH_FROM_TO</code> is equal to 1 we can rewrite <code>PATH_FROM_TO(index, 1, 0) = 1</code> to <code>PATH_FROM_TO(index, 1, 0) ≥ 1</code> . Therefore we simplify <u><code>PATH_FROM_TO</code></u> to <code>PATH_FROM_TO</code> .
Automaton	Figure 4.271 depicts the automaton associated to the <code>lex_less</code> constraint. Let VAR1_i and VAR2_i respectively be the <code>var</code> attributes of the i^{th} items of the <code>VECTOR1</code> and the <code>VECTOR2</code> collections. To each pair $(\text{VAR1}_i, \text{VAR2}_i)$ corresponds a signature variable S_i as well as the following signature constraint: $(\text{VAR1}_i < \text{VAR2}_i \Leftrightarrow S_i = 1) \wedge (\text{VAR1}_i = \text{VAR2}_i \Leftrightarrow S_i = 2) \wedge (\text{VAR1}_i > \text{VAR2}_i \Leftrightarrow S_i = 3)$.
Remark	A <i>multiset ordering</i> constraint and its corresponding filtering algorithm are described in [127].
Algorithm	The first complete filtering algorithm for this constraint was presented in [36]. A second complete filtering algorithm, detecting entailment in a more eager way, was given in [128]. This second algorithm was derived from a deterministic finite automata. A third complete filtering algorithm extending the algorithm presented in [36] detecting entailment is given in the PhD thesis of Z.Kızıltan [129, page 95]. The previous thesis [129, pages 105–109] presents also a filtering algorithm handling the fact that a given variable has more than one occurrence.
Used in	<code>lex_chain_less</code> .
See also	<code>lex_between</code> , <code>lex_lesseq</code> , <code>lex_greater</code> , <code>lex_greatereq</code> , <code>lex_chain_lesseq</code> .
Key words	order constraint, vector, symmetry, matrix symmetry, lexicographic order, multiset ordering, duplicated variables, Berge-acyclic constraint network, automaton, automaton without counters, derived collection.

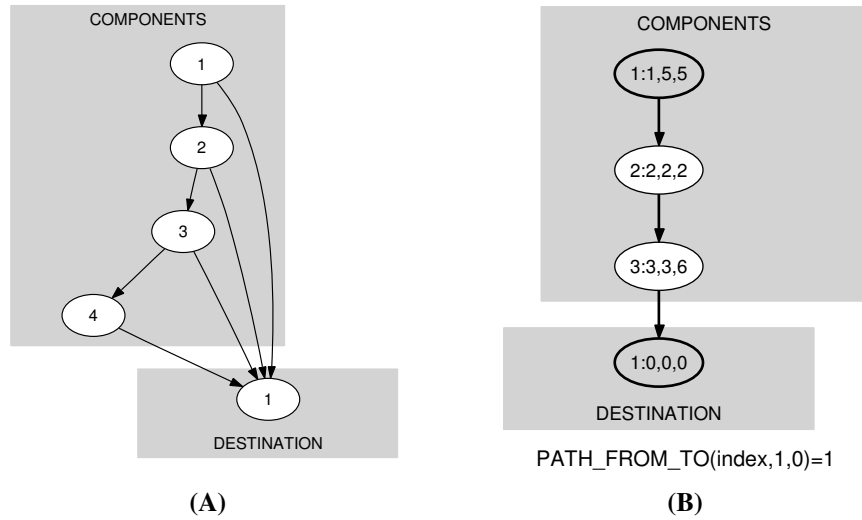


Figure 4.270: Initial and final graph of the `lex_less` constraint

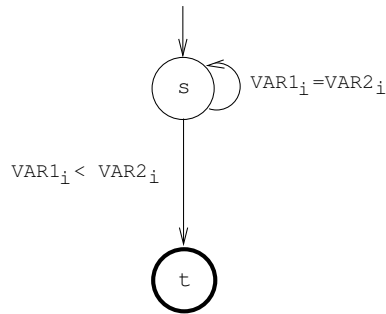


Figure 4.271: Automaton of the `lex_less` constraint

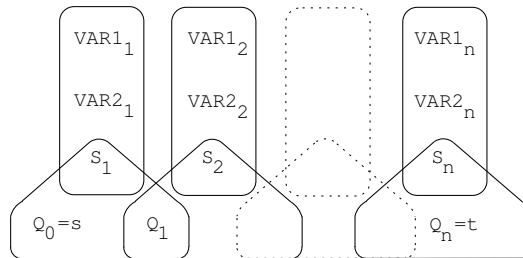


Figure 4.272: Hypergraph of the reformulation corresponding to the automaton of the `lex_less` constraint

20030820

609

4.124 lex_lesseq

Origin	CHIP
Constraint	<code>lex_lesseq(VECTOR1, VECTOR2)</code>
Argument(s)	VECTOR1 : <code>collection(var - dvar)</code> VECTOR2 : <code>collection(var - dvar)</code>
Restriction(s)	<code>required(VECTOR1, var)</code> <code>required(VECTOR2, var)</code> <code> VECTOR1 = VECTOR2 </code>
Purpose	VECTOR1 is lexicographically less than or equal to VECTOR2. Given two vectors, \vec{X} and \vec{Y} of n components, $\langle X_0, \dots, X_n \rangle$ and $\langle Y_0, \dots, Y_n \rangle$, \vec{X} is lexicographically less than or equal to \vec{Y} if and only if $n = 0$ or $X_0 < Y_0$ or $X_0 = Y_0$ and $\langle X_1, \dots, X_n \rangle$ is lexicographically less than or equal to $\langle Y_1, \dots, Y_n \rangle$.
Derived Collection(s)	$\text{col} \left(\begin{array}{l} \text{DESTINATION} - \text{collection}(\text{index} - \text{int}, \text{x} - \text{int}, \text{y} - \text{int}), \\ [\text{item}(\text{index} - 0, \text{x} - 0, \text{y} - 0)] \end{array} \right)$ <hr/> $\text{col} \left(\begin{array}{l} \text{COMPONENTS} - \text{collection}(\text{index} - \text{int}, \text{x} - \text{dvar}, \text{y} - \text{dvar}), \\ [\text{item}(\text{index} - \text{VECTOR1.key}, \text{x} - \text{VECTOR1.var}, \text{y} - \text{VECTOR2.var})] \end{array} \right)$
Arc input(s)	COMPONENTS DESTINATION
Arc generator	$\text{PRODUCT}(\text{PATH}, \text{VOID}) \mapsto \text{collection}(\text{item1}, \text{item2})$
Arc arity	2
Arc constraint(s)	$\bigvee \left(\begin{array}{l} \text{item2.index} > 0 \wedge \text{item1.x} = \text{item1.y}, \\ \text{item1.index} < \text{VECTOR1} \wedge \text{item2.index} = 0 \wedge \text{item1.x} < \text{item1.y}, \\ \text{item1.index} = \text{VECTOR1} \wedge \text{item2.index} = 0 \wedge \text{item1.x} \leq \text{item1.y} \end{array} \right)$
Graph property(ies)	$\text{PATH_FROM_TO}(\text{index}, 1, 0) = 1$
Example	$\text{lex_lesseq} \left(\begin{array}{l} \{\text{var} - 5, \text{var} - 2, \text{var} - 3, \text{var} - 1\}, \\ \{\text{var} - 5, \text{var} - 2, \text{var} - 6, \text{var} - 2\} \end{array} \right)$ $\text{lex_lesseq} \left(\begin{array}{l} \{\text{var} - 5, \text{var} - 2, \text{var} - 3, \text{var} - 9\}, \\ \{\text{var} - 5, \text{var} - 2, \text{var} - 3, \text{var} - 9\} \end{array} \right)$

Parts (A) and (B) of Figure 4.273 respectively show the initial and final graph associated to the first example. Since we use the **PATH_FROM_TO** graph property we show on the final graph the following information:

- The vertices which respectively correspond to the start and the end of the required path are stressed in bold.
- The arcs on the required path are also stressed in bold.

Graph model

The vertices of the initial graph are generated in the following way:

- We create a vertex c_i for each pair of components which both have the same index i .
- We create an additional dummy vertex called d .

The arcs of the initial graph are generated in the following way:

- We create an arc between c_i and d . When c_i was generated from the last components of both vectors We associate to this arc the arc constraint $\text{item}_1.x \leq \text{item}_2.y$; Otherwise we associate to this arc the arc constraint $\text{item}_1.x < \text{item}_2.y$;
- We create an arc between c_i and c_{i+1} . We associate to this arc the arc constraint $\text{item}_1.x = \text{item}_2.y$.

The `lex_lesseq` constraint holds when there exist a path from c_1 to d . This path can be interpreted as a maximum sequence of *equality* constraints on the prefix of both vectors, eventually followed by a *less than* constraint.

Signature

Since the maximum value returned by the graph property `PATH_FROM_TO` is equal to 1 we can rewrite `PATH_FROM_TO(index, 1, 0) = 1` to `PATH_FROM_TO(index, 1, 0) ≥ 1`. Therefore we simplify `PATH_FROM_TO` to `PATH_TO`.

Automaton

Figure 4.274 depicts the automaton associated to the `lex_lesseq` constraint. Let VAR1_i and VAR2_i respectively be the `var` attributes of the i^{th} items of the `VECTOR1` and the `VECTOR2` collections. To each pair $(\text{VAR1}_i, \text{VAR2}_i)$ corresponds a signature variable S_i as well as the following signature constraint: $(\text{VAR1}_i < \text{VAR2}_i \Leftrightarrow S_i = 1) \wedge (\text{VAR1}_i = \text{VAR2}_i \Leftrightarrow S_i = 2) \wedge (\text{VAR1}_i > \text{VAR2}_i \Leftrightarrow S_i = 3)$.

Remark

A *multiset ordering* constraint and its corresponding filtering algorithm are described in [127].

Algorithm

The first complete filtering algorithm for this constraint was presented in [36]. A second complete filtering algorithm, detecting entailment in a more eager way, was given in [128]. This second algorithm was derived from a deterministic finite automata. A third complete filtering algorithm extending the algorithm presented in [36] detecting entailment is given in the PhD thesis of Z.Kızıltan [129, page 95]. The previous thesis [129, pages 105–109] presents also a filtering algorithm handling the fact that a given variable has more than one occurrence.

Used in

`lex_between`, `lex_chain_lesseq`.

See also

`lex_less`, `lex_greater`, `lex_greatereq`, `lex_chain_less`.

Key words

order constraint, vector, symmetry, matrix symmetry, lexicographic order, multiset ordering, duplicated variables, Berge-acyclic constraint network, automaton, automaton without counters, derived collection.

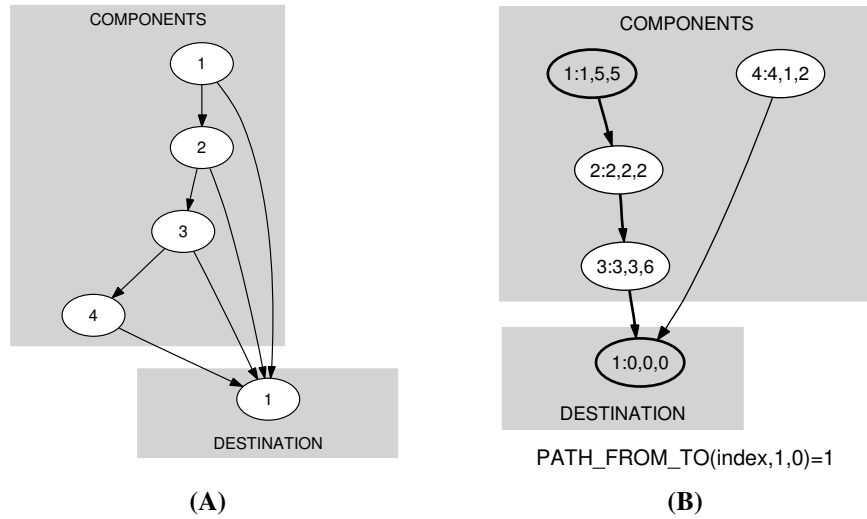


Figure 4.273: Initial and final graph of the *lex_lesseq* constraint

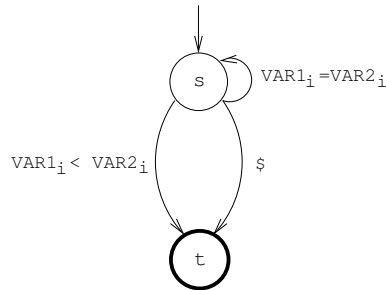


Figure 4.274: Automaton of the *lex_lesseq* constraint

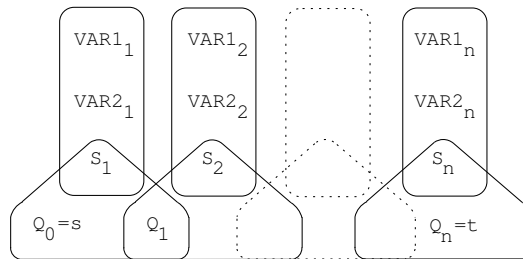


Figure 4.275: Hypergraph of the reformulation corresponding to the automaton of the *lex_lesseq* constraint

4.125 link_set_to_booleans

Origin	Inspired by <code>domain_constraint</code> .
Constraint	<code>link_set_to_booleans(SVAR, BOOLEANS)</code>
Argument(s)	<pre>SVAR : svar BOOLEANS : collection(bool - dvar, val - int)</pre>
Restriction(s)	<pre>required(BOOLEANS, [bool, val]) BOOLEANS.bool ≥ 0 BOOLEANS.bool ≤ 1 distinct(BOOLEANS, val)</pre>
Purpose	<p>Make the link between a set variable <code>SVAR</code> and those 0-1 variables that are associated to each potential value belonging to <code>SVAR</code>: The 0-1 variables, which are associated to a value belonging to the set variable <code>SVAR</code>, are equal to 1, while the remaining 0-1 variables are all equal to 0.</p>
Derived Collection(s)	<code>col(SET - collection(one - int, setvar - svar), [item(one - 1, setvar - SVAR)])</code>
Arc input(s)	SET BOOLEANS
Arc generator	<i>PRODUCT</i> \mapsto <code>collection(set, booleans)</code>
Arc arity	2
Arc constraint(s)	<code>booleans.bool = set.one \Leftrightarrow in_set(booleans.val, set.setvar)</code>
Graph property(ies)	<code>NARC = BOOLEANS </code>

Example	$\text{link_set_to_booleans} \left(\begin{array}{c} \{1, 3, 4\}, \\ \left(\begin{array}{cc} \text{bool} - 0 & \text{val} - 0, \\ \text{bool} - 1 & \text{val} - 1, \\ \text{bool} - 0 & \text{val} - 2, \\ \text{bool} - 1 & \text{val} - 3, \\ \text{bool} - 1 & \text{val} - 4, \\ \text{bool} - 0 & \text{val} - 5 \end{array} \right) \end{array} \right)$
----------------	---

In the previous example, the 0-1 variables associated to the values 1,3 and 4 are all set to 1, while the other 0-1 variables are set to 0. The `link_set_to_booleans` constraint holds since the final graph contains exactly 6 arcs (one for each 0-1 variable). Parts (A) and (B) of Figure 4.276 respectively show the initial and final graph. Since we use the `NARC` graph property, the arcs of the final graph are stressed in bold.

Graph model	The <code>link_set_to_booleans</code> constraint is modelled with the following bipartite graph. The first set of vertices corresponds to one single vertex containing the set variable. The second class of vertices contains one vertex for each item of the collection <code>BOOLEANS</code> . The arc constraint between the set variable <code>SVAR</code> and one potential value v of the set variable expresses the following:
--------------------	--

- If the 0-1 variable associated to v is equal to 1 then v should belong to SVAR.
- Otherwise if the 0-1 variable associated to v is equal to 0 then v should not belong to SVAR.

Since all arc constraints should hold the final graph contains exactly $|\text{BOOLEANS}|$ arcs.

Signature

Since the initial graph contains $|\text{BOOLEANS}|$ arcs the maximum number of arcs of the final graph is equal to $|\text{BOOLEANS}|$. Therefore we can rewrite the graph property $\text{NARC} = |\text{BOOLEANS}|$ to $\text{NARC} \geq |\text{BOOLEANS}|$ and simplify $\overline{\text{NARC}}$ to $\overline{\text{NARC}}$.

Usage

This constraint is used in order to make the link between a formulation using set variables and a formulation based on linear programming.

See also

`domain_constraint`, `clique`, `symmetric_gcc`, `tour`, `strongly_connected`, `path_from_to`.

Key words

decomposition, value constraint, channeling constraint, set channel, linear programming, constraint involving set variables, derived collection.

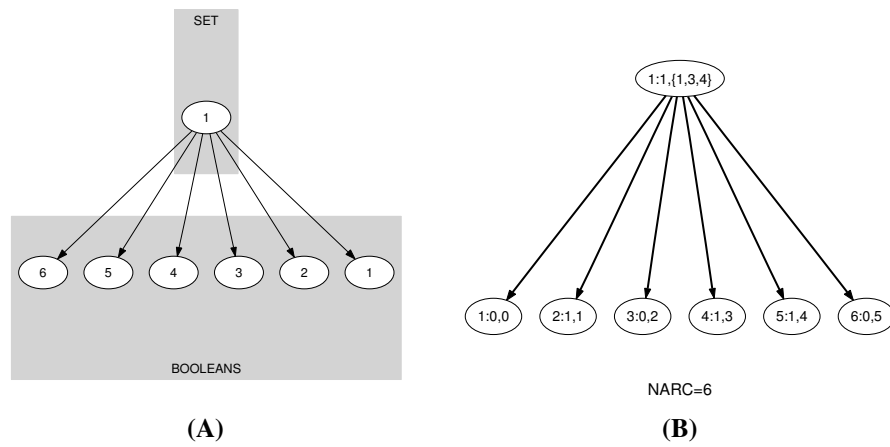


Figure 4.276: Initial and final graph of the link_set_to_booleans constraint

20030820

617

4.126 longest_change

Origin	Derived from change.
Constraint	longest_change(SIZE, VARIABLES, CTR)
Argument(s)	SIZE : dvar VARIABLES : collection(var – dvar) CTR : atom
Restriction(s)	SIZE ≥ 0 SIZE < VARIABLES required(VARIABLES, var) CTR ∈ [=, ≠, <, ≥, >, ≤]

Purpose

SIZE is the maximum number of consecutive variables of the collection VARIABLES for which constraint CTR holds in an uninterrupted way. We count a change when $X \text{ CTR } Y$ holds; X and Y are two consecutive variables of the collection VARIABLES.

Arc input(s)	VARIABLES
Arc generator	$PATH \mapsto \text{collection}(\text{variables1}, \text{variables2})$
Arc arity	2
Arc constraint(s)	variables1.var CTR variables2.var
Graph property(ies)	$MAX_NCC = SIZE$

Example

$$\text{longest_change} \left(4, \left\{ \begin{array}{l} \text{var} - 8, \\ \text{var} - 8, \\ \text{var} - 3, \\ \text{var} - 4, \\ \text{var} - 1, \\ \text{var} - 1, \\ \text{var} - 5, \\ \text{var} - 5, \\ \text{var} - 2 \end{array} \right\}, \neq \right)$$

Parts (A) and (B) of Figure 4.277 respectively show the initial and final graph. Since we use the **MAX_NCC** graph property we show the largest connected component of the final graph. It corresponds to the longest period of uninterrupted changes: Sequence 8, 3, 4, 1, which involves 4 consecutive variables.

Graph model

In order to specify the longest_change constraint, we use **MAX_NCC**, which is the number of vertices of the largest connected component. Since the initial graph corresponds to a path, this will be the length of the longest path in the final graph.

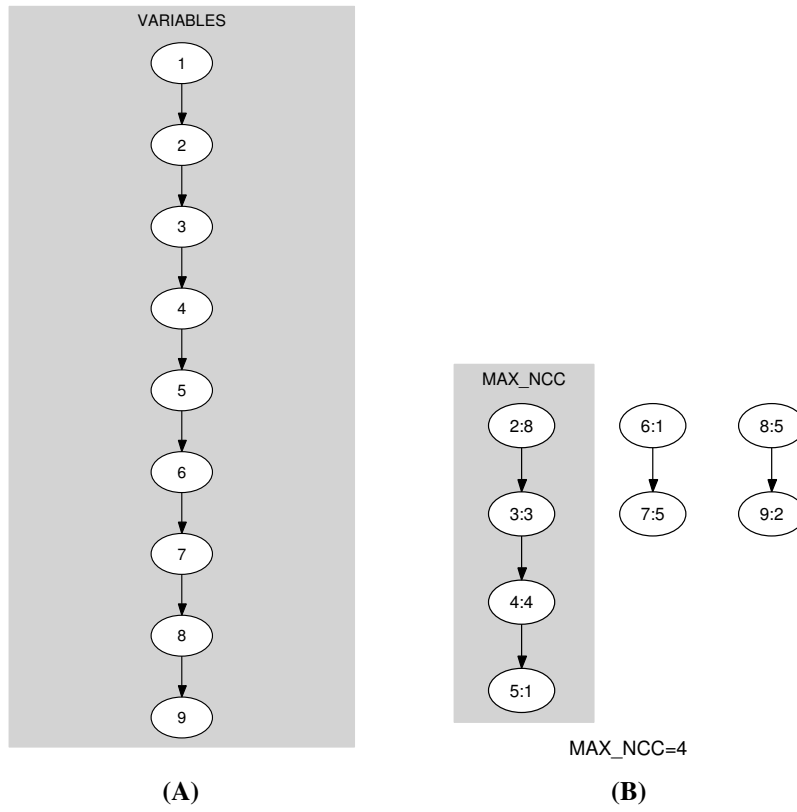


Figure 4.277: Initial and final graph of the longest_change constraint

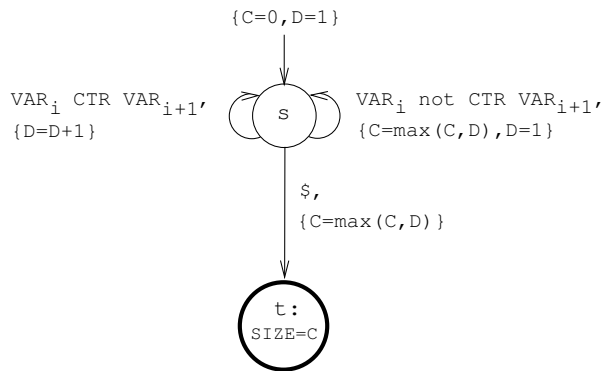


Figure 4.278: Automaton of the longest_change constraint

Automaton

Figure 4.278 depicts the automaton associated to the `longest_change` constraint. To each pair of consecutive variables (VAR_i, VAR_{i+1}) of the collection `VARIABLES` corresponds a 0-1 signature variable S_i . The following signature constraint links VAR_i, VAR_{i+1} and S_i : $VAR_i \text{ CTR } VAR_{i+1} \Leftrightarrow S_i$.

See also

`change`.

Key words

timetabling constraint, automaton, automaton with counters, sliding cyclic(1) constraint network(3).

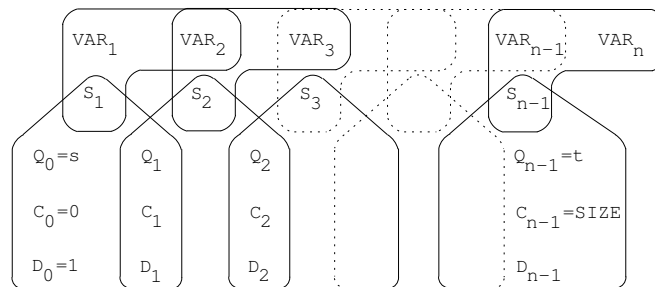


Figure 4.279: Hypergraph of the reformulation corresponding to the automaton of the `longest_change` constraint

20000128

621

4.127 map

Origin	Inspired by [130]
Constraint	map(NBCYCLE, NBTREE, NODES)
Argument(s)	NBCYCLE : dvar NBTREE : dvar NODES : collection(index - int, succ - dvar)
Restriction(s)	NBCYCLE ≥ 0 NBTREE ≥ 0 required(NODES, [index, succ]) NODES.index ≥ 1 NODES.index $\leq NODES $ distinct(NODES, index) NODES.succ ≥ 1 NODES.succ $\leq NODES $

Purpose

Number of trees and number of cycles of a map. We take the description of a map from [130, page 459]:

Every map decomposes into a set of connected components, also called connected maps. Each component consists of the set of all points that wind up on the same cycle, with each point on the cycle attached to a tree of all points that enter the cycle at that point.

Arc input(s)	NODES
Arc generator	<i>CLIQUE</i> \mapsto collection(nodes1, nodes2)
Arc arity	2
Arc constraint(s)	nodes1.succ = nodes2.index
Graph property(ies)	<ul style="list-style-type: none"> • NCC = NBCYCLE • NTREE = NBTREE

Example

$$\text{map} \left(2, 3, \left\{ \begin{array}{l} \text{index} - 1 \quad \text{succ} - 5, \\ \text{index} - 2 \quad \text{succ} - 9, \\ \text{index} - 3 \quad \text{succ} - 8, \\ \text{index} - 4 \quad \text{succ} - 2, \\ \text{index} - 5 \quad \text{succ} - 9, \\ \text{index} - 6 \quad \text{succ} - 2, \\ \text{index} - 7 \quad \text{succ} - 9, \\ \text{index} - 8 \quad \text{succ} - 8, \\ \text{index} - 9 \quad \text{succ} - 1 \end{array} \right\} \right)$$

Parts (A) and (B) of Figure 4.280 respectively show the initial and final graph. Since we use the NCC graph property, we display the two connected components of the

final graph. Each of them corresponds to a connected map. The first connected map is made up from one circuit and two trees, while the second one consists of one circuit and one tree. Since we also use the **NTREE** graph property, we display with a double circle those vertices which do not belong to any circuit but for which at least one successor belong to a circuit.

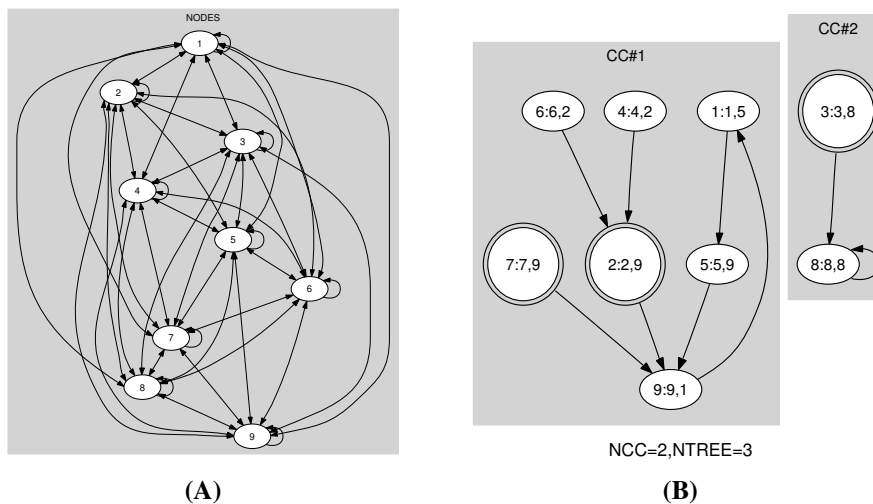


Figure 4.280: Initial and final graph of the map constraint

Graph model

Observe that, for the argument **NBTREE** of the map constraint, we consider a definition different from the one used for the argument **NTREES** of the **tree** constraint:

- In the **map** constraint the number of trees **NBTREE** is equal to the number of vertices of the final graph, which both do not belong to any circuit and have a successor which is located on a circuit. Therefore we count three trees in the previous example.
- In the **tree** constraint the number of trees **NTREES** is equal to the number of connected components of the final graph.

See also

cycle, tree, graph_crossing.

Key words

graph constraint, graph partitioning constraint, connected component.

4.128 max_index

Origin	N. Beldiceanu
Constraint	<code>max_index(MAX_INDEX, VARIABLES)</code>
Argument(s)	<code>MAX_INDEX</code> : dvar <code>VARIABLES</code> : <code>collection(index – int, var – dvar)</code>
Restriction(s)	<code> VARIABLES > 0</code> <code>MAX_INDEX ≥ 0</code> <code>MAX_INDEX ≤ VARIABLES </code> <code>required(VARIABLES, [index, var])</code> <code>VARIABLES.index ≥ 1</code> <code>VARIABLES.index ≤ VARIABLES </code> <code>distinct(VARIABLES, index)</code>
Purpose	<code>MAX_INDEX</code> is the index of the variables corresponding to the maximum value of the collection of variables <code>VARIABLES</code> .
Arc input(s)	<code>VARIABLES</code>
Arc generator	<code>CLIQUE</code> \mapsto <code>collection(variables1, variables2)</code>
Arc arity	2
Arc constraint(s)	<code>variables1.key = variables2.key \vee variables1.var > variables2.var</code>
Graph property(ies)	<code>ORDER(0, 0, index) = MAX_INDEX</code>
Example	$\max_index \left(3, \left\{ \begin{array}{l} \text{index} - 1 \quad \text{var} - 3, \\ \text{index} - 2 \quad \text{var} - 2, \\ \text{index} - 3 \quad \text{var} - 7, \\ \text{index} - 4 \quad \text{var} - 2, \\ \text{index} - 5 \quad \text{var} - 6 \end{array} \right\} \right)$ <p>Parts (A) and (B) of Figure 4.281 respectively show the initial and final graph. Since we use the ORDER graph property, the vertex of rank 0 (without considering the loops) of the final graph is shown in gray.</p>
Automaton	Figure 4.282 depicts the automaton associated to the <code>max_index</code> constraint. To each item of the collection <code>VARIABLES</code> corresponds a signature variable S_i , which is equal to 0.
See also	<code>min_index</code> .
Key words	order constraint, maximum, automaton, automaton with counters, alpha-acyclic constraint network(4).

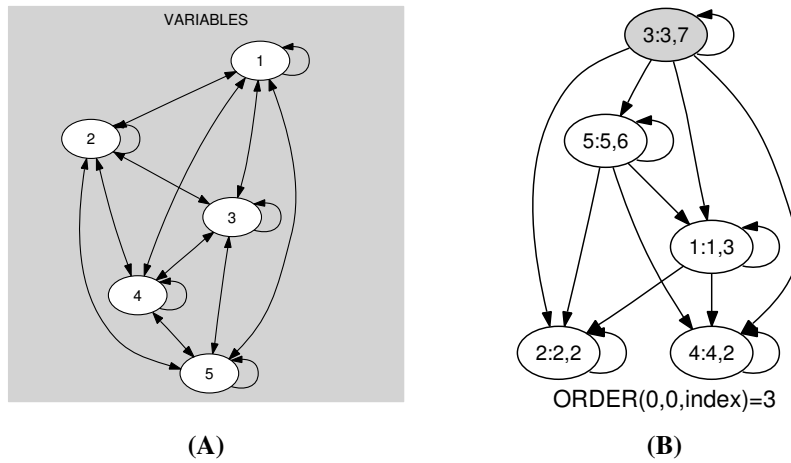


Figure 4.281: Initial and final graph of the max_index constraint

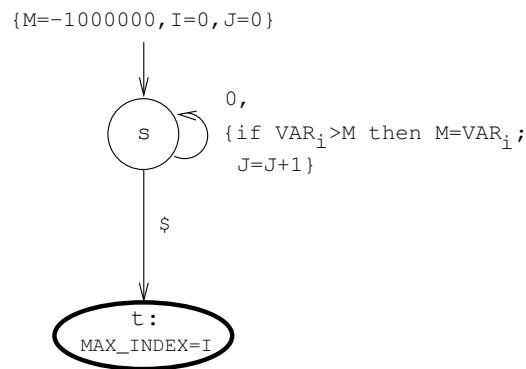


Figure 4.282: Automaton of the max_index constraint

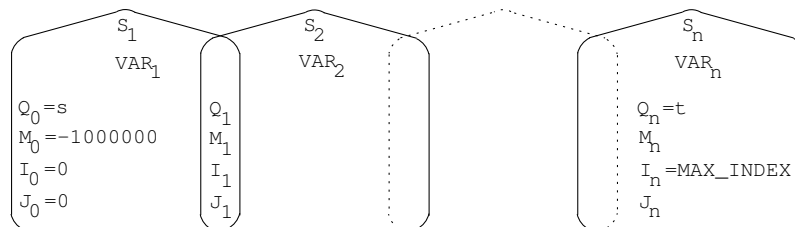


Figure 4.283: Hypergraph of the reformulation corresponding to the automaton of the max_index constraint

4.129 max_n

Origin	[33]
Constraint	<code>max_n(MAX, RANK, VARIABLES)</code>
Argument(s)	<pre> MAX : dvar RANK : int VARIABLES : collection(var - dvar) </pre>
Restriction(s)	<pre> VARIABLES > 0 RANK ≥ 0 RANK < VARIABLES required(VARIABLES, var) </pre>
Purpose	<p>MAX is the maximum value of rank RANK (i.e. the RANKth largest distinct value) of the collection of domain variables VARIABLES. Sources have a rank of 0.</p>
<hr/>	
Arc input(s)	VARIABLES
Arc generator	<i>CLIQUE</i> \mapsto <code>collection(variables1, variables2)</code>
Arc arity	2
Arc constraint(s)	<code>variables1.key = variables2.key \vee variables1.var > variables2.var</code>
Graph property(ies)	<code>ORDER(RANK, MININT, var) = MAX</code>
<hr/>	
Example	$\max_n \left(6, 1, \left\{ \begin{array}{l} \text{var} - 3, \\ \text{var} - 1, \\ \text{var} - 7, \\ \text{var} - 1, \\ \text{var} - 6 \end{array} \right\} \right)$ <p>Parts (A) and (B) of Figure 4.284 respectively show the initial and final graph. Since we use the ORDER graph property, the vertex of rank 1 (without considering the loops) of the final graph is shown in gray.</p>
Algorithm	[33].
See also	<code>maximum, min_n</code> .
Key words	order constraint, rank, maximum.

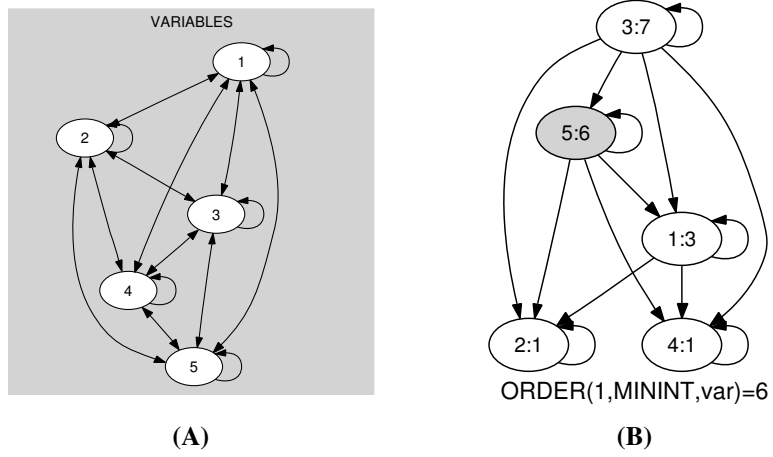


Figure 4.284: Initial and final graph of the max_n constraint

4.130 max_nvalue

Origin	Derived from <code>nvalue</code> .
Constraint	<code>max_nvalue(MAX, VARIABLES)</code>
Argument(s)	<code>MAX</code> : <code>dvar</code> <code>VARIABLES</code> : <code>collection(var - dvar)</code>
Restriction(s)	$MAX \geq 1$ $MAX \leq VARIABLES $ <code>required(VARIABLES, var)</code>
Purpose	<code>MAX</code> is the maximum number of times that the same value is taken by the variables of the collection <code>VARIABLES</code> .
Arc input(s)	<code>VARIABLES</code>
Arc generator	<code>CLIQUE</code> \mapsto <code>collection(variables1, variables2)</code>
Arc arity	2
Arc constraint(s)	<code>variables1.var = variables2.var</code>
Graph property(ies)	<code>MAX_NSCC = MAX</code>

Example	$\text{max_nvalue} \left(3, \left\{ \begin{array}{l} \text{var} - 9, \\ \text{var} - 1, \\ \text{var} - 7, \\ \text{var} - 1, \\ \text{var} - 1, \\ \text{var} - 6, \\ \text{var} - 7, \\ \text{var} - 7, \\ \text{var} - 4, \\ \text{var} - 9 \end{array} \right\} \right)$
----------------	--

In the previous example, values 1, 4, 6, 7, 9 are respectively used 3, 1, 1, 3, 2 times. So the maximum number of time `MAX` that a same value occurs is 3. Parts (A) and (B) of Figure 4.285 respectively show the initial and final graph. Since we use the `MAX_NSCC` graph property, we show the largest strongly connected component of the final graph.

Graph model	Because of the arc constraint, each strongly connected component of the final graph corresponds to a distinct value which is assigned to a subset of variables of the <code>VARIABLES</code> collection. Therefore the number of vertices of the largest strongly connected component is equal to the mostly used value.
--------------------	--

Automaton	Figure 4.286 depicts the automaton associated to the <code>max_nvalue</code> constraint. To each item of the collection <code>VARIABLES</code> corresponds a signature variable S_i , which is equal to 0.
------------------	--

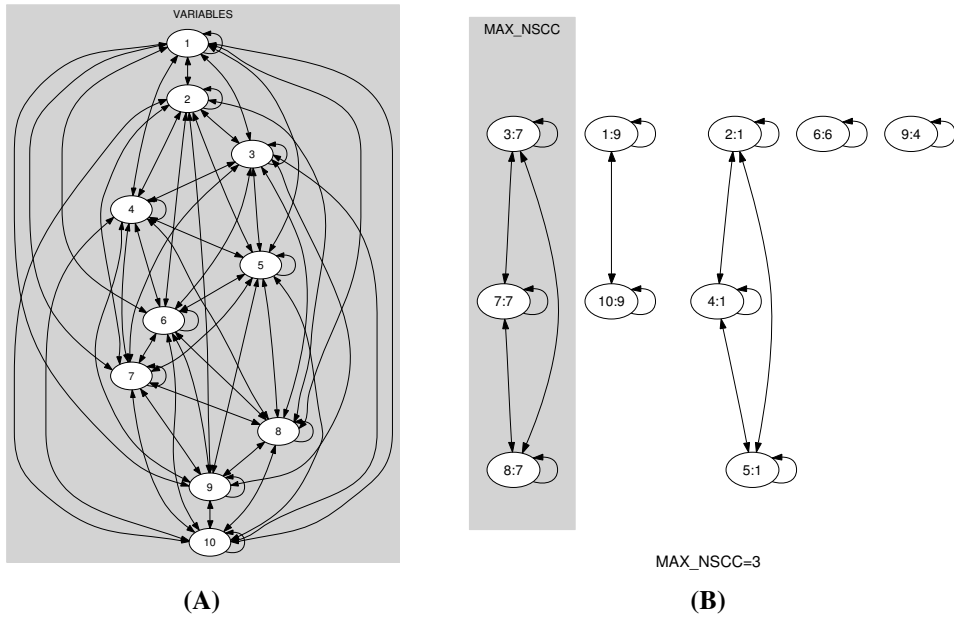


Figure 4.285: Initial and final graph of the max_nvalue constraint

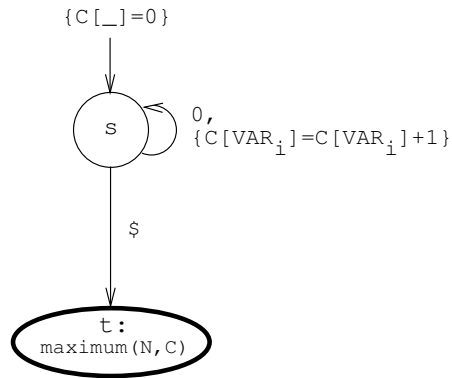


Figure 4.286: Automaton of the max_nvalue constraint

- Usage** This constraint may be used in order to replace a set of `count` or `among` constraints were one would have to generate explicitly one constraint for each potential value. Also useful for constraining the number of occurrences of the mostly used value without knowing this value in advance and without giving explicitly an upper limit on the number of occurrences of each value as it is done in the `global_cardinality` constraint.
- See also** `nvalue`, `min_nvalue`.
- Key words** value constraint, assignment, maximum number of occurrences, maximum, automaton, automaton with array of counters, equivalence.

20000128

631

4.131 max_size_set_of_consecutive_var

Origin	N. Beldiceanu
Constraint	<code>max_size_set_of_consecutive_var(MAX, VARIABLES)</code>
Argument(s)	<code>MAX</code> : dvar <code>VARIABLES</code> : collection(var – dvar)
Restriction(s)	$MAX \geq 1$ $MAX \leq VARIABLES $ <code>required(VARIABLES, var)</code>
Purpose	<code>MAX</code> is the size of the largest set of variables of the collection <code>VARIABLES</code> which all take their value in a set of consecutive values.
Arc input(s)	<code>VARIABLES</code>
Arc generator	<code>CLIQUE</code> \mapsto <code>collection(variables1, variables2)</code>
Arc arity	2
Arc constraint(s)	<code>abs(variables1.var – variables2.var) \leq 1</code>
Graph property(ies)	<code>MAX_NSCC = MAX</code>

Example	<code>max_size_set_of_consecutive_var</code> $6, \left(\begin{array}{c} \left(\begin{array}{c} \text{var} - 3, \\ \text{var} - 1, \\ \text{var} - 3, \\ \text{var} - 7, \\ \text{var} - 4, \\ \text{var} - 1, \\ \text{var} - 2, \\ \text{var} - 8, \\ \text{var} - 7, \\ \text{var} - 6 \end{array} \right) \end{array} \right)$
----------------	---

In the previous example, the following sets of variables $\{\text{var} - 3, \text{var} - 1, \text{var} - 3, \text{var} - 4, \text{var} - 1, \text{var} - 2\}$ and $\{\text{var} - 7, \text{var} - 8, \text{var} - 7, \text{var} - 6\}$ take their values in the two following sets of consecutive values $\{1, 2, 3, 4\}$ and $\{6, 7, 8\}$. The `max_size_set_of_consecutive_var` constraint holds since the cardinality of the largest set of variables is 6. Parts (A) and (B) of Figure 4.287 respectively show the initial and final graph. Since we use the `MAX_NSCC` graph property, we show the largest strongly connected component of the final graph.

Graph model	Since the arc constraint is symmetric each strongly connected component of the final graph corresponds exactly to one connected component of the final graph.
See also	<code>nset_of_consecutive_values</code> .
Key words	value constraint, consecutive values, maximum.

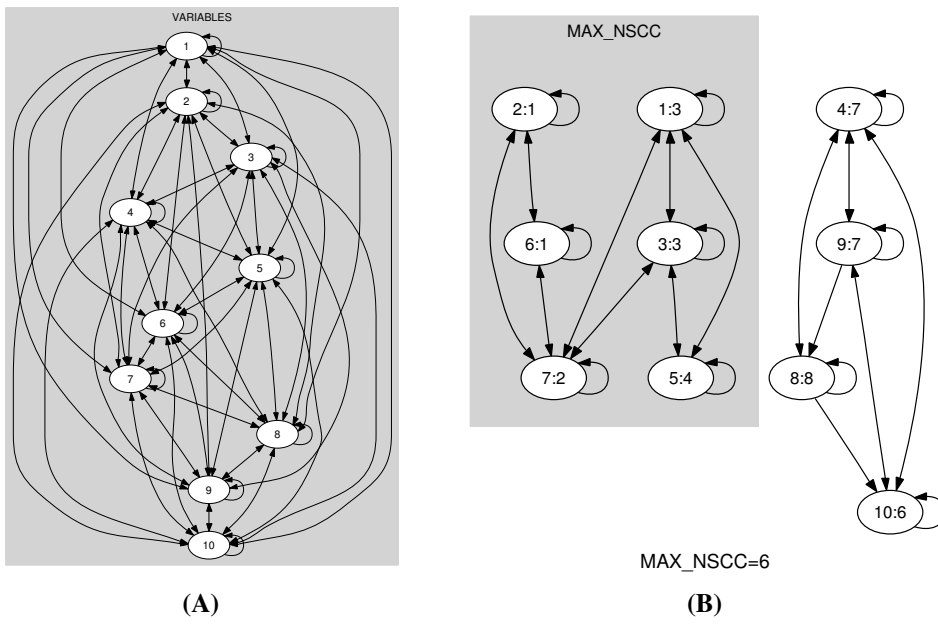


Figure 4.287: Initial and final graph of the `max_size_set_of_consecutive_var` constraint

4.132 maximum

Origin	CHIP
Constraint	maximum(MAX, VARIABLES)
Argument(s)	MAX : dvar VARIABLES : collection(var – dvar)
Restriction(s)	VARIABLES > 0 required(VARIABLES, var)
Purpose	MAX is the maximum value of the collection of domain variables VARIABLES.
<hr/>	
Arc input(s)	VARIABLES
Arc generator	<i>CLIQUE</i> \mapsto collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	variables1.key = variables2.key \vee variables1.var > variables2.var
Graph property(ies)	ORDER (0, MININT, var) = MAX
Example	$\text{maximum} \left(7, \left\{ \begin{array}{l} \text{var} - 3, \\ \text{var} - 2, \\ \text{var} - 7, \\ \text{var} - 2, \\ \text{var} - 6 \end{array} \right\} \right)$ <p>Parts (A) and (B) of Figure 4.288 respectively show the initial and final graph. Since we use the ORDER graph property, the vertex of rank 0 (without considering the loops) of the final graph is shown in gray.</p>
Graph model	We use a similar definition that the one that was utilized for the minimum constraint. Within the arc constraint, we replace the comparison operator < by >.
Automaton	Figure 4.289 depicts the automaton associated to the maximum constraint. Let VAR _{<i>i</i>} be the <i>i</i> th variable of the VARIABLES collection. To each pair (MAX, VAR _{<i>i</i>}) corresponds a signature variable S _{<i>i</i>} as well as the following signature constraint: (MAX > VAR _{<i>i</i>} \Leftrightarrow S _{<i>i</i>} = 0) \wedge (MAX = VAR _{<i>i</i>} \Leftrightarrow S _{<i>i</i>} = 1) \wedge (MAX < VAR _{<i>i</i>} \Leftrightarrow S _{<i>i</i>} = 2).
Usage	In some project scheduling problems one has to introduce dummy activities which correspond for instance to the completion time of a given set of activities. In this context one can use the maximum constraint to get the maximum end of a set of tasks.
Remark	Note that maximum is a constraint and not just a function that computes the maximum value of a collection of variables: The values of MAX influence the variables and reciprocally the values of the variables influence MAX.

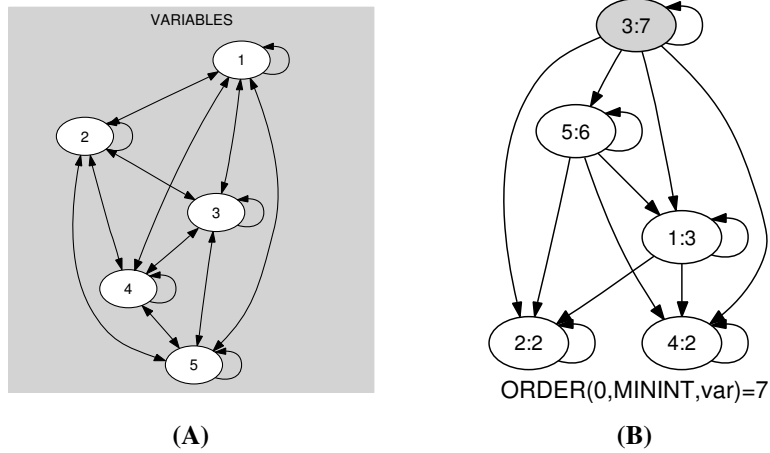


Figure 4.288: Initial and final graph of the maximum constraint

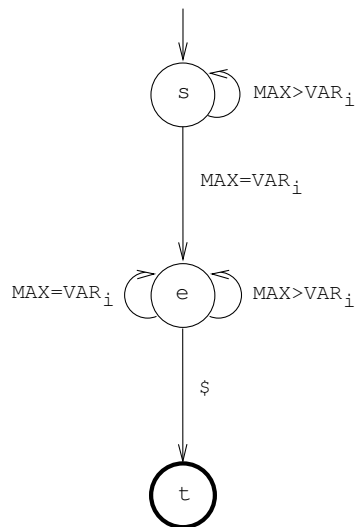


Figure 4.289: Automaton of the maximum constraint

Algorithm	[33].
See also	minimum.
Key words	order constraint, maximum, automaton, automaton without counters, centered cyclic(1) constraint network(1).

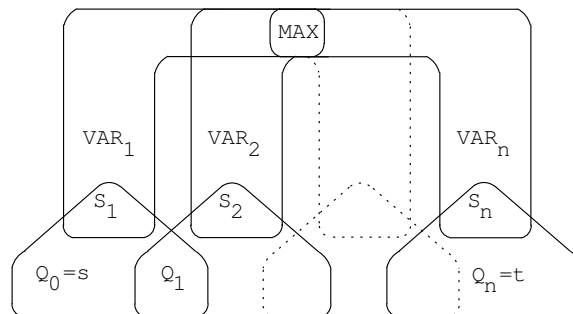


Figure 4.290: Hypergraph of the reformulation corresponding to the automaton of the maximum constraint

20000128

637

4.133 maximum_modulo

Origin	Derived from maximum.
Constraint	maximum_modulo(MAX, VARIABLES, M)
Argument(s)	MAX : dvar VARIABLES : collection(var - dvar) M : int
Restriction(s)	VARIABLES > 0 M > 0 required(VARIABLES, var)
Purpose	MAX is a maximum value of the collection of domain variables VARIABLES according to the following partial ordering: $(X \bmod M) < (Y \bmod M)$.
Arc input(s)	VARIABLES
Arc generator	<i>CLIQUE</i> \mapsto collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	variables1.key = variables2.key \vee variables1.var mod M > variables2.var mod M
Graph property(ies)	ORDER (0, MININT, var) = MAX
Example	$\text{maximum_modulo} \left(5, \left\{ \begin{array}{l} \text{var} - 9, \\ \text{var} - 1, \\ \text{var} - 7, \\ \text{var} - 6, \\ \text{var} - 5 \end{array} \right\}, 3 \right)$ <p>Parts (A) and (B) of Figure 4.291 respectively show the initial and final graph. Since we use the ORDER graph property, the vertex of rank 0 (without considering the loops) of the final graph is shown in gray.</p>
See also	maximum, minimum_modulo.
Key words	order constraint, modulo, maximum.

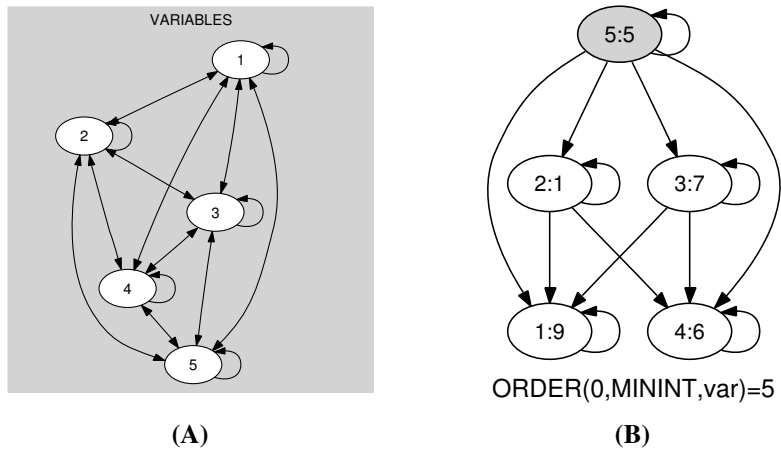


Figure 4.291: Initial and final graph of the `maximum_modulo` constraint

4.134 min_index

Origin	N. Beldiceanu
Constraint	<code>min_index(MIN_INDEX, VARIABLES)</code>
Argument(s)	<p><code>MIN_INDEX</code> : dvar</p> <p><code>VARIABLES</code> : <code>collection(index – int, var – dvar)</code></p>
Restriction(s)	<p><code> VARIABLES > 0</code></p> <p><code>MIN_INDEX ≥ 0</code></p> <p><code>MIN_INDEX ≤ VARIABLES </code></p> <p><code>required(VARIABLES, [index, var])</code></p> <p><code>VARIABLES.index ≥ 1</code></p> <p><code>VARIABLES.index ≤ VARIABLES </code></p> <p><code>distinct(VARIABLES, index)</code></p>
Purpose	<p><code>MIN_INDEX</code> is the index of the variables corresponding to the minimum value of the collection of variables <code>VARIABLES</code>.</p>
Arc input(s)	<code>VARIABLES</code>
Arc generator	<code>CLIQUE</code> \mapsto <code>collection(variables1, variables2)</code>
Arc arity	2
Arc constraint(s)	<code>variables1.key = variables2.key \vee variables1.var < variables2.var</code>
Graph property(ies)	<code>ORDER(0, 0, index) = MIN_INDEX</code>

Example	$\min_index \left(2, \left\{ \begin{array}{l} \text{index} - 1 \quad \text{var} - 3, \\ \text{index} - 2 \quad \text{var} - 2, \\ \text{index} - 3 \quad \text{var} - 7, \\ \text{index} - 4 \quad \text{var} - 2, \\ \text{index} - 5 \quad \text{var} - 6 \end{array} \right\} \right)$ $\min_index \left(4, \left\{ \begin{array}{l} \text{index} - 1 \quad \text{var} - 3, \\ \text{index} - 2 \quad \text{var} - 2, \\ \text{index} - 3 \quad \text{var} - 7, \\ \text{index} - 4 \quad \text{var} - 2, \\ \text{index} - 5 \quad \text{var} - 6 \end{array} \right\} \right)$
----------------	---

Parts (A) and (B) of Figure 4.292 respectively show the initial and final graph associated to both examples. Since we use the **ORDER** graph property, the vertices of rank 0 (without considering the loops) of the final graph are shown in gray.

Graph model	<p>Within the context of scheduling, assume the variables of the <code>VARIABLES</code> collection correspond to the starts of a set of tasks. Then <code>MIN_INDEX</code> gives the indexes of those tasks which can be scheduled first.</p>
--------------------	---

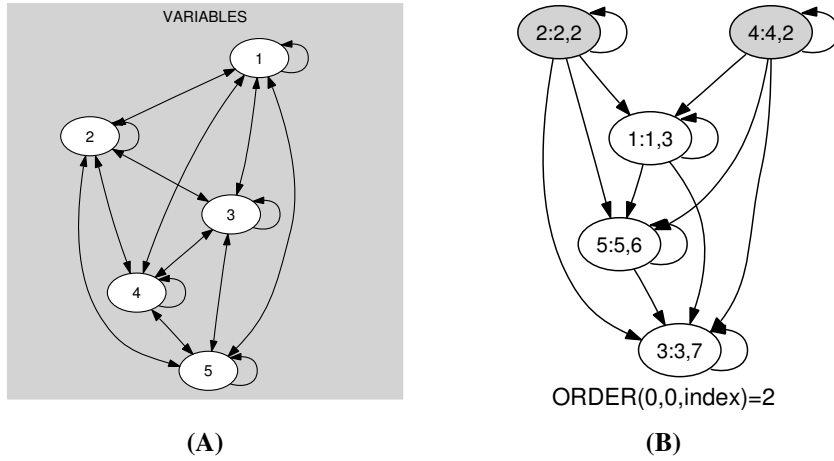


Figure 4.292: Initial and final graph of the min_index constraint

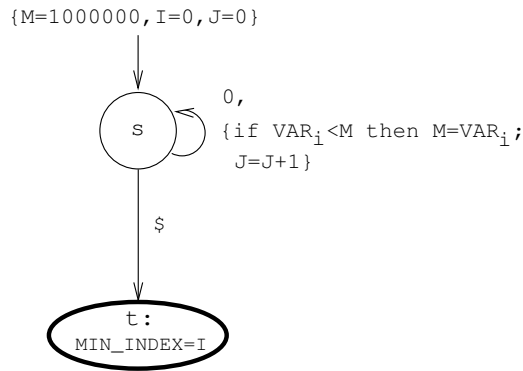


Figure 4.293: Automaton of the min_index constraint

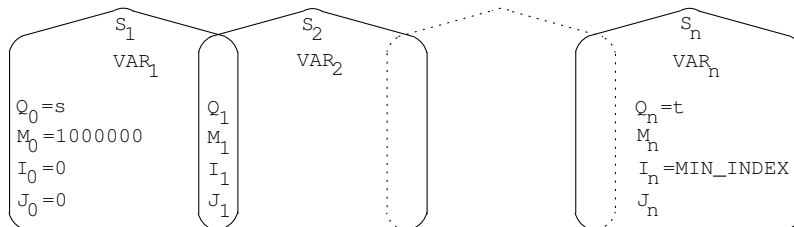


Figure 4.294: Hypergraph of the reformulation corresponding to the automaton of the min_index constraint

- Automaton** Figure 4.293 depicts the automaton associated to the `min_index` constraint. Figure 4.293 depicts the automaton associated to the `min_index` constraint. To each item of the collection `VARIABLES` corresponds a signature variable S_i , which is equal to 0.
- See also** `max_index`.
- Key words** order constraint, minimum, automaton, automaton with counters, alpha-acyclic constraint network(4).

20030820

643

4.135 min_n

Origin	[33]
Constraint	<code>min_n(MIN, RANK, VARIABLES)</code>
Argument(s)	<pre> MIN : dvar RANK : int VARIABLES : collection(var - dvar) </pre>
Restriction(s)	<pre> VARIABLES > 0 RANK ≥ 0 RANK < VARIABLES required(VARIABLES, var) </pre>
Purpose	<p>MIN is the minimum value of rank RANK (i.e. the $RANK^{th}$ smallest distinct value) of the collection of domain variables VARIABLES. Sources have a rank of 0.</p>
Arc input(s)	VARIABLES
Arc generator	<code>CLIQUE</code> \mapsto <code>collection(variables1, variables2)</code>
Arc arity	2
Arc constraint(s)	<code>variables1.key = variables2.key \vee variables1.var < variables2.var</code>
Graph property(ies)	<code>ORDER(RANK, MAXINT, var) = MIN</code>
Example	$\min_n \left(3, 1, \left\{ \begin{array}{l} \text{var} - 3, \\ \text{var} - 1, \\ \text{var} - 7, \\ \text{var} - 1, \\ \text{var} - 6 \end{array} \right\} \right)$ <p>Note that identical values are only counted once. This is why the minimum of order 1 is 3 instead of 1 in the previous example. Parts (A) and (B) of Figure 4.295 respectively show the initial and final graph. Since we use the ORDER graph property, the vertex of rank 1 (without considering the loops) of the final graph is shown in gray.</p>
Graph model	A generalization of the minimum constraint.
Automaton	Figure 4.296 depicts the automaton associated to the <code>min_n</code> constraint. Figure 4.296 depicts the automaton associated to the <code>min_n</code> constraint. To each item of the collection VARIABLES corresponds a signature variable S_i , which is equal to 1.
Algorithm	[33].
See also	<code>minimum</code> , <code>max_n</code> , <code>ith_pos_different_from_0</code> .
Key words	order constraint, rank, minimum, maxint, automaton, automaton with array of counters.

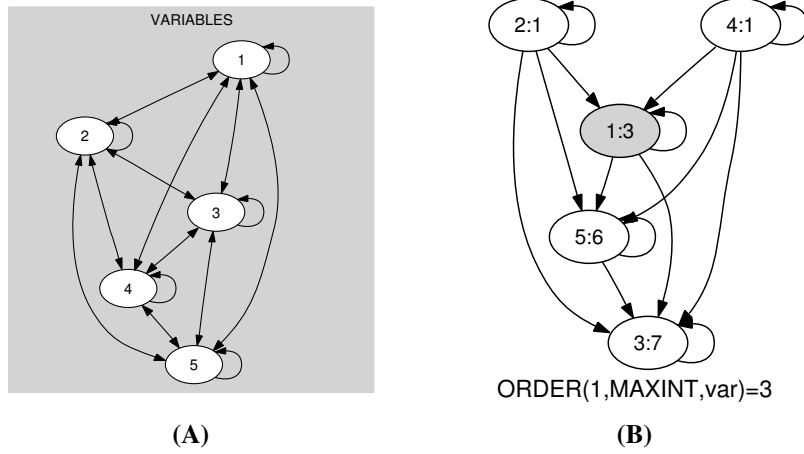


Figure 4.295: Initial and final graph of the min_n constraint

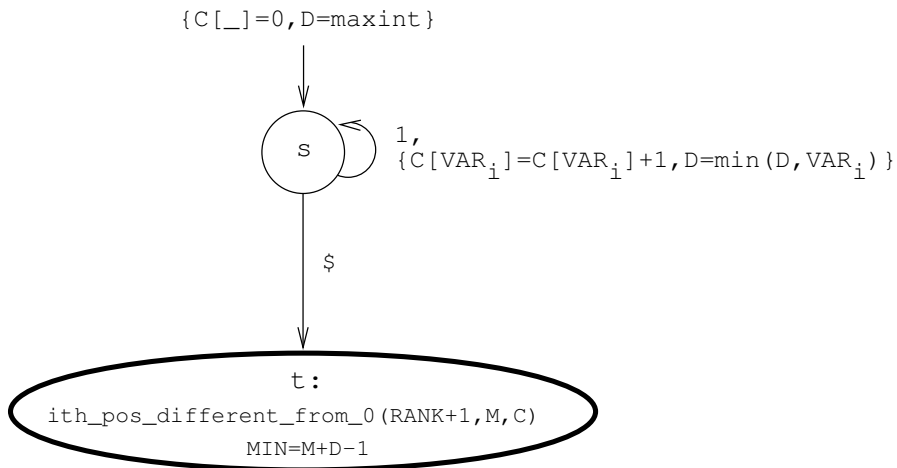


Figure 4.296: Automaton of the min_n constraint

4.136 min_nvalue

Origin	N. Beldiceanu
Constraint	<code>min_nvalue(MIN, VARIABLES)</code>
Argument(s)	<pre> MIN : dvar VARIABLES : collection(var - dvar) </pre>
Restriction(s)	<pre> MIN ≥ 1 MIN ≤ VARIABLES required(VARIABLES, var) </pre>
Purpose	<p>MIN is the minimum number of times that the same value is taken by the variables of the collection VARIABLES.</p>
Arc input(s)	VARIABLES
Arc generator	<i>CLIQUE</i> \mapsto <code>collection(variables1, variables2)</code>
Arc arity	2
Arc constraint(s)	<code>variables1.var = variables2.var</code>
Graph property(ies)	<code>MIN_NSCC = MIN</code>
Example	$\text{min_nvalue} \left(2, \left\{ \begin{array}{l} \text{var} - 9, \\ \text{var} - 1, \\ \text{var} - 7, \\ \text{var} - 1, \\ \text{var} - 1, \\ \text{var} - 7, \\ \text{var} - 7, \\ \text{var} - 7, \\ \text{var} - 7, \\ \text{var} - 9 \end{array} \right\} \right)$ <p>In the previous example, values 1, 7, 9 are respectively used 3, 5, 2 times. So the minimum number of time that a same value occurs is 2. Parts (A) and (B) of Figure 4.297 respectively show the initial and final graph. Since we use the <code>MIN_NSCC</code> graph property, we show the smallest strongly connected component of the final graph.</p>
Automaton	Figure 4.298 depicts the automaton associated to the <code>min_nvalue</code> constraint. To each item of the collection VARIABLES corresponds a signature variable S_i , which is equal to 0.
Usage	This constraint may be used in order to replace a set of <code>count</code> or <code>among</code> constraints were one would have to generate explicitly one constraint for each potential value. Also useful for constraining the number of occurrences of the less used value without knowing this value in advance and without giving explicitly a lower limit on the number of occurrences of each value as it is done in the <code>global_cardinality</code> constraint.

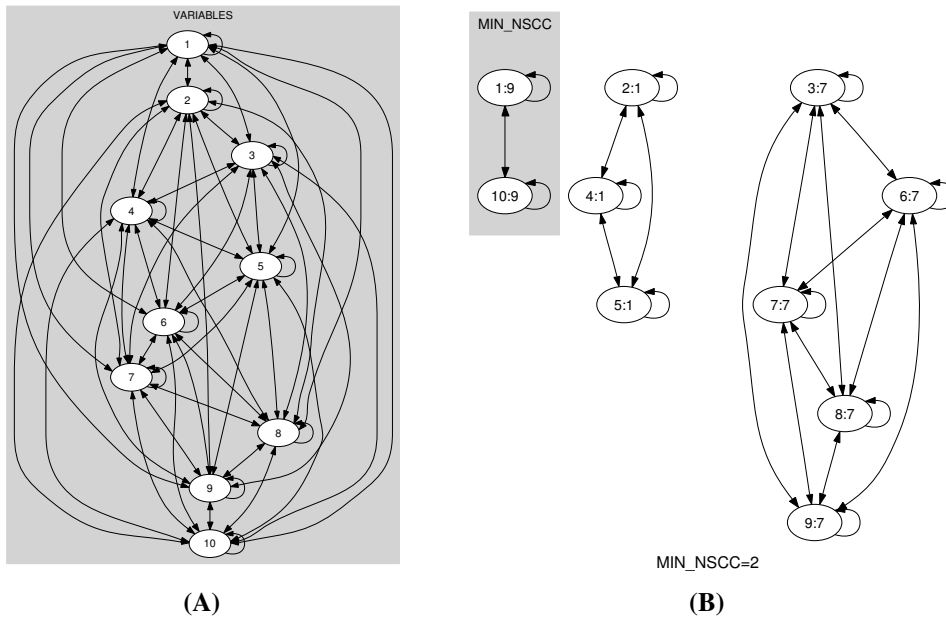


Figure 4.297: Initial and final graph of the min_nvalue constraint

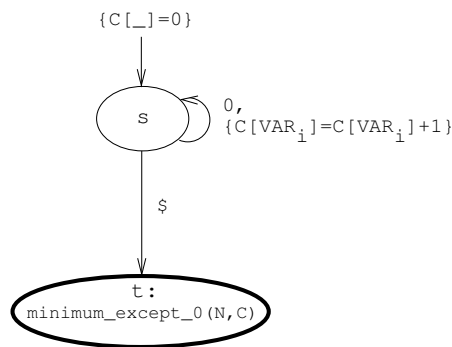


Figure 4.298: Automaton of the min_nvalue constraint

See also nvalue, max_nvalue.

Key words value constraint, assignment, minimum number of occurrences, minimum, automaton, automaton with array of counters, equivalence.

20000128

649

4.137 min_size_set_of_consecutive_var

Origin	N. Beldiceanu
Constraint	<code>min_size_set_of_consecutive_var(MIN, VARIABLES)</code>
Argument(s)	<code>MIN</code> : dvar <code>VARIABLES</code> : collection(var – dvar)
Restriction(s)	$MIN \geq 1$ $MIN \leq VARIABLES $ <code>required(VARIABLES, var)</code>
Purpose	<code>MIN</code> is the size of the smallest set of variables of the collection <code>VARIABLES</code> which all take their value in a set of consecutive values.
Arc input(s)	<code>VARIABLES</code>
Arc generator	<code>CLIQUE</code> \mapsto <code>collection(variables1, variables2)</code>
Arc arity	2
Arc constraint(s)	<code>abs(variables1.var – variables2.var) \leq 1</code>
Graph property(ies)	<code>MIN_NSCC = MIN</code>
Example	<code>min_size_set_of_consecutive_var</code> $4, \left\{ \begin{array}{l} \text{var} - 3, \\ \text{var} - 1, \\ \text{var} - 3, \\ \text{var} - 7, \\ \text{var} - 4, \\ \text{var} - 1, \\ \text{var} - 2, \\ \text{var} - 8, \\ \text{var} - 7, \\ \text{var} - 6 \end{array} \right\}$
	<p>In the previous example, the following sets of variables $\{\text{var} - 3, \text{var} - 1, \text{var} - 3, \text{var} - 4, \text{var} - 1, \text{var} - 2\}$ and $\{\text{var} - 7, \text{var} - 8, \text{var} - 7, \text{var} - 6\}$ take their values in the two following sets of consecutive values $\{1, 2, 3, 4\}$ and $\{6, 7, 8\}$. The <code>min_size_set_of_consecutive_var</code> constraint holds since the cardinality of the smallest set of variables is 4. Parts (A) and (B) of Figure 4.299 respectively show the initial and final graph. Since we use the <code>MIN_NSCC</code> graph property, we show the smallest strongly connected component of the final graph.</p>
Graph model	Since the arc constraint is symmetric each strongly connected component of the final graph corresponds exactly to one connected component of the final graph.
See also	<code>nset_of_consecutive_values</code> .
Key words	value constraint, assignment, consecutive values, minimum.

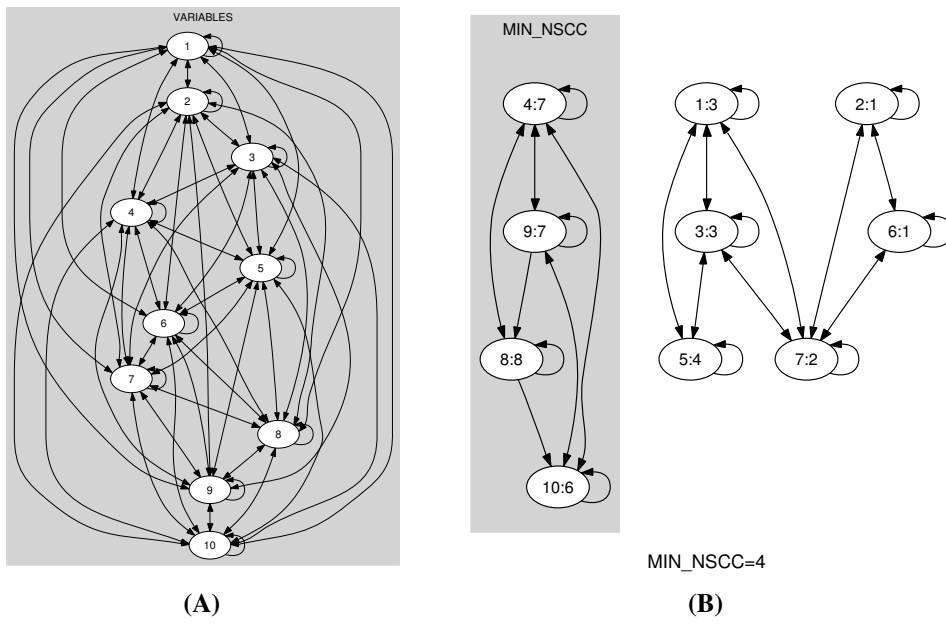


Figure 4.299: Initial and final graph of the min_size_set_of_consecutive_var constraint

4.138 minimum

Origin	CHIP
Constraint	<code>minimum(MIN, VARIABLES)</code>
Argument(s)	<code>MIN</code> : <code>dvar</code> <code>VARIABLES</code> : <code>collection(var - dvar)</code>
Restriction(s)	$ VARIABLES > 0$ <code>required(VARIABLES, var)</code>
Purpose	<code>MIN</code> is the minimum value of the collection of domain variables <code>VARIABLES</code> .
<hr/>	
Arc input(s)	<code>VARIABLES</code>
Arc generator	<code>CLIQUE</code> \mapsto <code>collection(variables1, variables2)</code>
Arc arity	2
Arc constraint(s)	<code>variables1.key = variables2.key \vee variables1.var < variables2.var</code>
Graph property(ies)	<code>ORDER(0, MAXINT, var) = MIN</code>
Example	$\text{minimum} \left(2, \left\{ \begin{array}{l} \text{var} - 3, \\ \text{var} - 2, \\ \text{var} - 7, \\ \text{var} - 2, \\ \text{var} - 6 \end{array} \right\} \right)$ <p>Parts (A) and (B) of Figure 4.300 respectively show the initial and final graph. Since we use the ORDER graph property, the vertices of rank 0 (without considering the loops) of the final graph are shown in gray.</p>
Graph model	The condition <code>variables1.key = variables2.key</code> holds if and only if <code>variables1</code> and <code>variables2</code> corresponds to the same vertex. It is used in order to enforce to keep all the vertices of the initial graph. <code>ORDER(0, MAXINT, var)</code> refers to the source vertices of the graph, i.e. those vertices that do not have any predecessor.
Automaton	Figure 4.301 depicts the automaton associated to the <code>minimum</code> constraint. Let VAR_i be the i^{th} variable of the <code>VARIABLES</code> collection. To each pair (MIN, VAR_i) corresponds a signature variable S_i as well as the following signature constraint: $(MIN < VAR_i \Leftrightarrow S_i = 0) \wedge (MIN = VAR_i \Leftrightarrow S_i = 1) \wedge (MIN > VAR_i \Leftrightarrow S_i = 2)$.
Remark	Note that <code>minimum</code> is a constraint and not just a function that computes the minimum value of a collection of variables: The values of <code>MIN</code> influence the variables and reciprocally the values of the variables influence <code>MIN</code> .
Algorithm	[33].

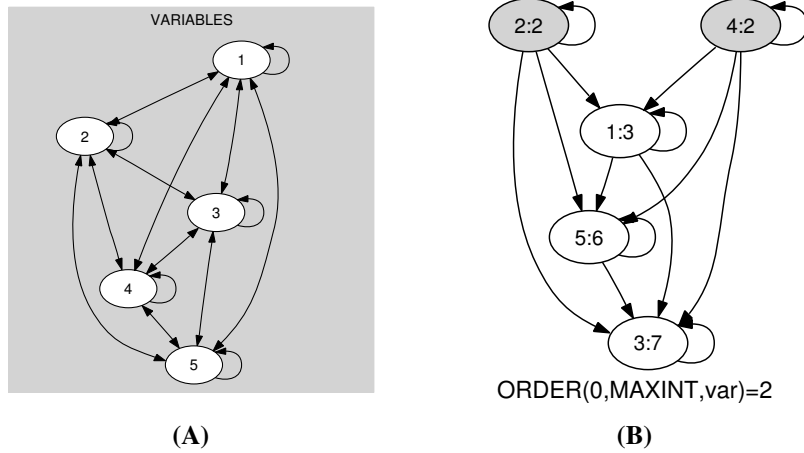


Figure 4.300: Initial and final graph of the minimum constraint

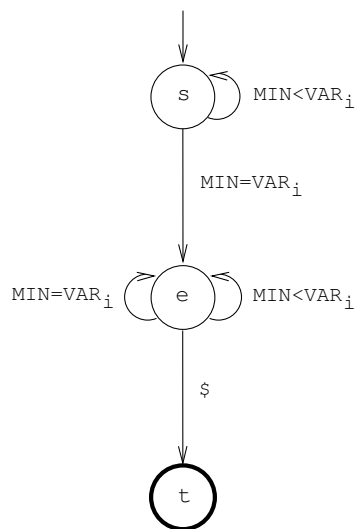


Figure 4.301: Automaton of the minimum constraint

Used in `minimum_greater_than`, `next_element`, `next_greater_element`.

See also `maximum`.

Key words order constraint, minimum, maxint, automaton, automaton without counters, centered cyclic(1) constraint network(1).

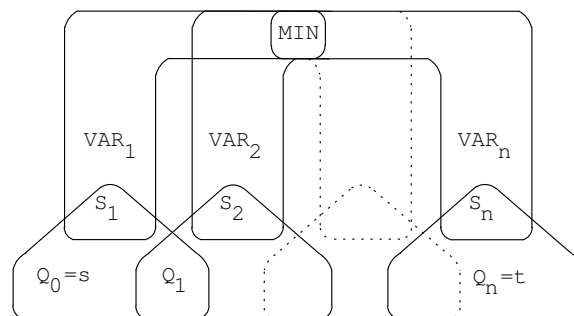


Figure 4.302: Hypergraph of the reformulation corresponding to the automaton of the minimum constraint

20000128

655

4.139 minimum_except_0

Origin	Derived from minimum.
Constraint	<code>minimum_except_0(MIN, VARIABLES)</code>
Argument(s)	<p><code>MIN</code> : dvar</p> <p><code>VARIABLES</code> : collection(var – dvar)</p>
Restriction(s)	<p><code> VARIABLES > 0</code></p> <p><code>required(VARIABLES, var)</code></p> <p><code>VARIABLES.var ≥ 0</code></p>
Purpose	<p>MIN is the minimum value of the collection of domain variables VARIABLES, ignoring all variables that take 0 as value.</p>
Arc input(s)	VARIABLES
Arc generator	<i>CLIQUE</i> \mapsto <code>collection(variables1, variables2)</code>
Arc arity	2
Arc constraint(s)	<ul style="list-style-type: none"> • <code>variables1.var ≠ 0</code> • <code>variables2.var ≠ 0</code> • <code>variables1.key = variables2.key ∨ variables1.var < variables2.var</code>
Graph property(ies)	<code>ORDER(0, MAXINT, var) = MIN</code>

Example	$\text{minimum_except_0} \left(3, \left\{ \begin{array}{l} \text{var} - 3, \\ \text{var} - 7, \\ \text{var} - 6, \\ \text{var} - 7, \\ \text{var} - 4, \\ \text{var} - 7 \end{array} \right\} \right)$
	$\text{minimum_except_0} \left(2, \left\{ \begin{array}{l} \text{var} - 3, \\ \text{var} - 2, \\ \text{var} - 0, \\ \text{var} - 7, \\ \text{var} - 2, \\ \text{var} - 6 \end{array} \right\} \right)$
	$\text{minimum_except_0} \left(1000000, \left\{ \begin{array}{l} \text{var} - 0, \\ \text{var} - 0, \\ \text{var} - 0, \\ \text{var} - 0, \\ \text{var} - 0 \end{array} \right\} \right)$

Parts (A) and (B) of Figure 4.303 respectively show the initial and final graph of

the second example. Since we use the **ORDER** graph property, the vertices of rank 0 (without considering the loops) of the final graph are shown in gray.

Since the graph associated to the third example does not contain any vertex, **ORDER** returns the default value **MAXINT**.

Graph model

Because of the first two conditions of the arc constraint, all vertices that correspond to 0 will be removed from the final graph.

Automaton

Figure 4.304 depicts the automaton associated to the `minimum_except_0` constraint. Let VAR_i be the i^{th} variable of the **VARIABLES** collection. To each pair $(\text{MIN}, \text{VAR}_i)$ corresponds a signature variable S_i as well as the following signature constraint:

$$((\text{VAR}_i = 0) \wedge (\text{MIN} \neq \text{MAXINT})) \Leftrightarrow S_i = 0 \wedge$$

$$((\text{VAR}_i = 0) \wedge (\text{MIN} = \text{MAXINT})) \Leftrightarrow S_i = 1 \wedge$$

$$((\text{VAR}_i \neq 0) \wedge (\text{MIN} = \text{VAR}_i)) \Leftrightarrow S_i = 2 \wedge$$

$$((\text{VAR}_i \neq 0) \wedge (\text{MIN} < \text{VAR}_i)) \Leftrightarrow S_i = 3.$$

Remark

The joker value 0 makes sense only because we restrict the variables of the **VARIABLES** collection to take non-negative values.

See also

`minimum`, `min_nvalue`.

Key words

order constraint, joker value, minimum, maxint, automaton, automaton without counters, centered cyclic(1) constraint network(1).

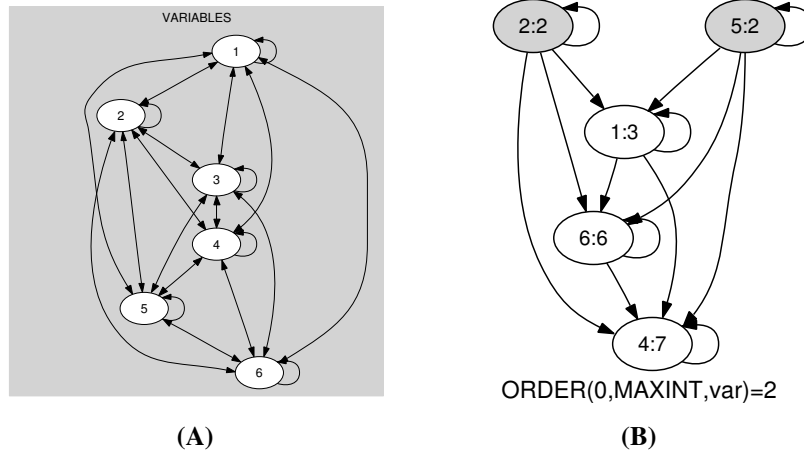


Figure 4.303: Initial and final graph of the `minimum_except_0` constraint

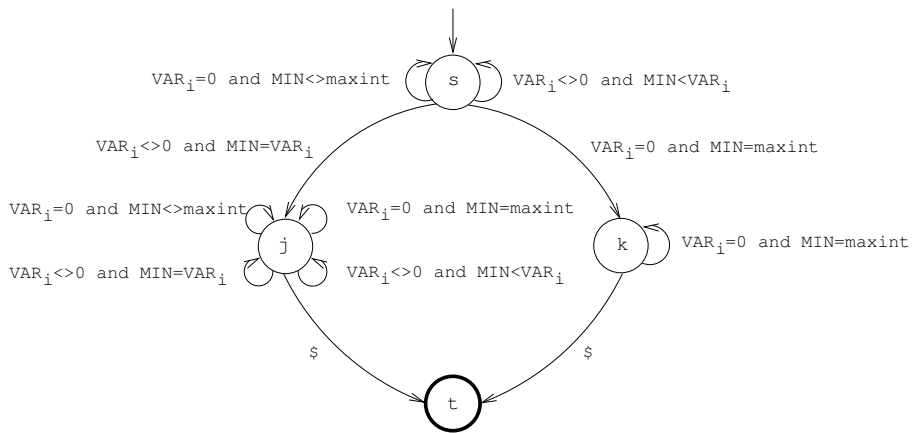


Figure 4.304: Automaton of the `minimum_except_0` constraint

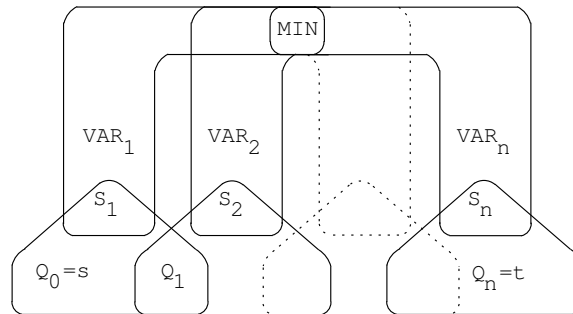


Figure 4.305: Hypergraph of the reformulation corresponding to the automaton of the `minimum_except_0` constraint

20030820

659

4.140 `minimum_greater_than`

Origin	N. Beldiceanu
Constraint	<code>minimum_greater_than(VAR1, VAR2, VARIABLES)</code>
Argument(s)	<code>VAR1</code> : dvar <code>VAR2</code> : dvar <code>VARIABLES</code> : <code>collection(var - dvar)</code>
Restriction(s)	<code> VARIABLES > 0</code> <code>required(VARIABLES, var)</code>
Purpose	<p><code>VAR1</code> is the smallest value strictly greater than <code>VAR2</code> of the collection of variables <code>VARIABLES</code>: This concretely means that there exist at least one variable of <code>VARIABLES</code> which take a value strictly greater than <code>VAR1</code>.</p>
Derived Collection(s)	<code>col(ITEM - collection(var - dvar), [item(var - VAR2)])</code>
Arc input(s)	ITEM VARIABLES
Arc generator	<i>PRODUCT</i> \mapsto <code>collection(item, variables)</code>
Arc arity	2
Arc constraint(s)	<code>item.var < variables.var</code>
Graph property(ies)	$\overline{\text{NARC}} > 0$
Sets	<i>SUCC</i> \mapsto <code>[source, variables]</code>
Constraint(s) on sets	<code>minimum(VAR1, variables)</code>
Example	$\text{minimum_greater_than} \left(5, 3, \left\{ \begin{array}{l} \text{var} - 8, \\ \text{var} - 5, \\ \text{var} - 3, \\ \text{var} - 8 \end{array} \right\} \right)$ <p>The <code>minimum_greater_than</code> constraint holds since value 5 is the smallest value strictly greater than value 3 among values 8, 5, 3 and 8. Parts (A) and (B) of Figure 4.306 respectively show the initial and final graph. Since we use the $\overline{\text{NARC}}$ graph property, the arcs of the final graph are stressed in bold. The source and the sinks of the final graph respectively correspond to the variable <code>VAR2</code> and to the variables of the <code>VARIABLES</code> collection which are strictly greater than <code>VAR2</code>. <code>VAR1</code> is set to the smallest value of the <code>var</code> attribute of the sinks of the final graph.</p>
Graph model	Similar to the <code>next_greater_element</code> constraint, except that there is no order on the variables of the collection <code>VARIABLES</code> .

Automaton

Figure 4.307 depicts the automaton associated to the `minimum_greater_than` constraint. Let VAR_i be the i^{th} variable of the `VARIABLES` collection. To each triple $(\text{VAR1}, \text{VAR2}, \text{VAR}_i)$ corresponds a signature variable S_i as well as the following signature constraint:

$$((\text{VAR}_i < \text{VAR1}) \wedge (\text{VAR}_i \leq \text{VAR2})) \Leftrightarrow S_i = 0 \wedge$$

$$((\text{VAR}_i = \text{VAR1}) \wedge (\text{VAR}_i \leq \text{VAR2})) \Leftrightarrow S_i = 1 \wedge$$

$$((\text{VAR}_i > \text{VAR1}) \wedge (\text{VAR}_i \leq \text{VAR2})) \Leftrightarrow S_i = 2 \wedge$$

$$((\text{VAR}_i < \text{VAR1}) \wedge (\text{VAR}_i > \text{VAR2})) \Leftrightarrow S_i = 3 \wedge$$

$$((\text{VAR}_i = \text{VAR1}) \wedge (\text{VAR}_i > \text{VAR2})) \Leftrightarrow S_i = 4 \wedge$$

$$((\text{VAR}_i > \text{VAR1}) \wedge (\text{VAR}_i > \text{VAR2})) \Leftrightarrow S_i = 5.$$

The automaton is constructed in order to fulfill the following conditions:

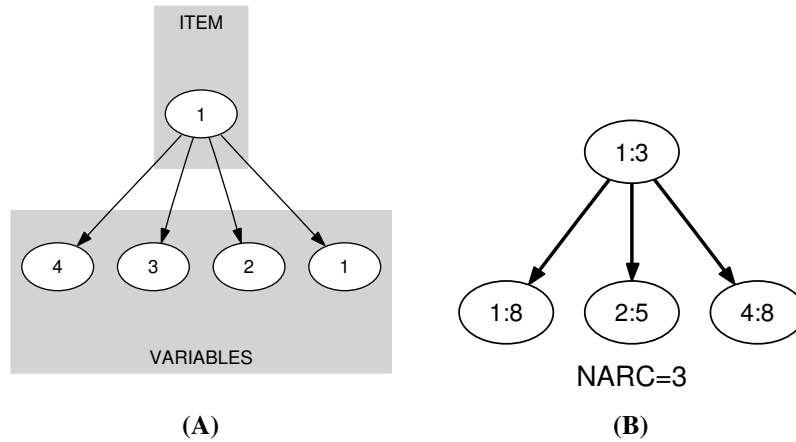
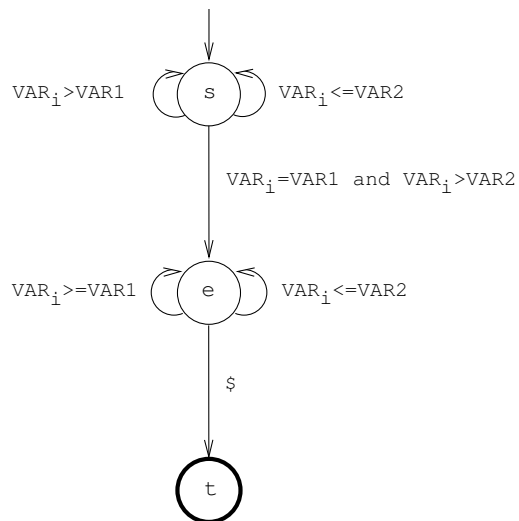
- We look for an item of the `VARIABLES` collection such that $\text{var}_i = \text{VAR1}$ and $\text{var}_i > \text{VAR2}$,
- There should not exist any item of the `VARIABLES` collection such that $\text{var}_i < \text{VAR1}$ and $\text{var}_i > \text{VAR2}$.

See also

`next_greater_element`.

Key words

order constraint, minimum, automaton, automaton without counters, centered cyclic(2) constraint network(1), derived collection.

Figure 4.306: Initial and final graph of the `minimum_greater_than` constraintFigure 4.307: Automaton of the `minimum_greater_than` constraint

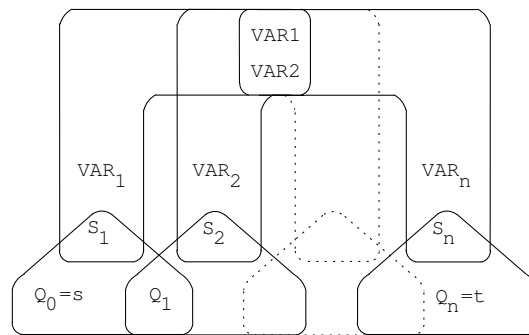


Figure 4.308: Hypergraph of the reformulation corresponding to the automaton of the `minimum_greater_than` constraint

4.141 minimum_modulo

Origin	Derived from <code>minimum</code> .
Constraint	<code>minimum_modulo(MIN, VARIABLES, M)</code>
Argument(s)	<pre> MIN : dvar VARIABLES : collection(var - dvar) M : int </pre>
Restriction(s)	<pre> VARIABLES > 0 M > 0 required(VARIABLES, var) </pre>
Purpose	<p><code>MIN</code> is a minimum value of the collection of domain variables <code>VARIABLES</code> according to the following partial ordering: $(X \bmod M) < (Y \bmod M)$.</p>
Arc input(s)	<code>VARIABLES</code>
Arc generator	<code>CLIQUE</code> \mapsto <code>collection(variables1, variables2)</code>
Arc arity	2
Arc constraint(s)	<code>variables1.key = variables2.key \vee variables1.var mod M < variables2.var mod M</code>
Graph property(ies)	<code>ORDER(0, MAXINT, var) = MIN</code>
Example	$\text{minimum_modulo} \left(6, \left\{ \begin{array}{l} \text{var} - 9, \\ \text{var} - 1, \\ \text{var} - 7, \\ \text{var} - 6, \\ \text{var} - 5 \end{array} \right\}, 3 \right)$ $\text{minimum_modulo} \left(9, \left\{ \begin{array}{l} \text{var} - 9, \\ \text{var} - 1, \\ \text{var} - 7, \\ \text{var} - 6, \\ \text{var} - 5 \end{array} \right\}, 3 \right)$
	<p>Parts (A) and (B) of Figure 4.309 respectively show the initial and final graph associated to the second example. Since we use the ORDER graph property, the vertex of rank 0 (without considering the loops) associated to value 9 is shown in gray.</p>
Graph model	We use a similar definition that the one that was utilized for the <code>minimum</code> constraint. Within the arc constraint we replace the condition $X < Y$ by the condition $(X \bmod M) < (Y \bmod M)$.
See also	<code>minimum</code> , <code>maximum_modulo</code> .
Key words	order constraint, modulo, maxint, minimum.

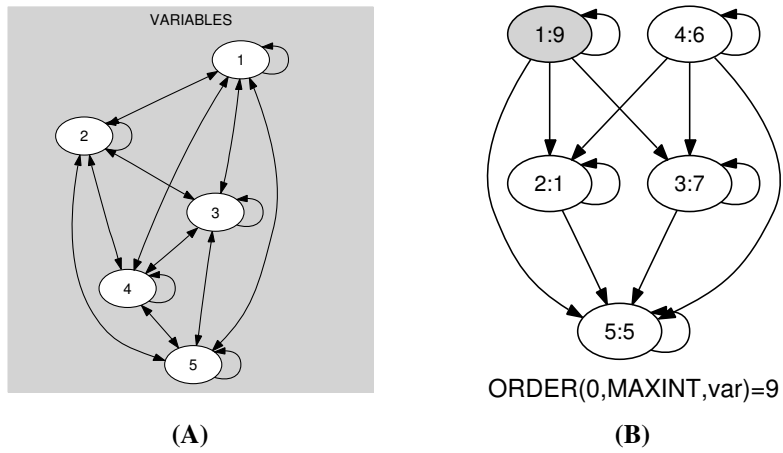


Figure 4.309: Initial and final graph of the minimum_modulo constraint

4.142 minimum_weight_alldifferent

Origin	[131]
Constraint	minimum_weight_alldifferent(VARIABLES, MATRIX, COST)
Synonym(s)	minimum_weight_alldiff, minimum_weight_alldistinct, min_weight_alldiff, min_weight_alldifferent, min_weight_alldistinct.
Argument(s)	VARIABLES : collection(var – dvar) MATRIX : collection(i – int, j – int, c – int) COST : dvar
Restriction(s)	$ VARIABLES > 0$ required(VARIABLES, var) $VARIABLES.var \geq 1$ $VARIABLES.var \leq VARIABLES $ required(MATRIX, [i, j, c]) increasing_seq(MATRIX, [i, j]) $MATRIX.i \geq 1$ $MATRIX.i \leq VARIABLES $ $MATRIX.j \geq 1$ $MATRIX.j \leq VARIABLES $ $ MATRIX = VARIABLES * VARIABLES $
Purpose	All variables of the VARIABLES collection should take a distinct value located within interval [1, VARIABLES]. In addition COST is equal to the sum of the costs associated to the fact that we assign value i to variable j . These costs are given by the matrix MATRIX.
Arc input(s)	VARIABLES
Arc generator	<i>CLIQUE</i> \mapsto collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	variables1.var = variables2.key
Graph property(ies)	<ul style="list-style-type: none"> • NTREE = 0 • <u>SUM_WEIGHT_ARC</u>(MATRIX[(variables1.key – 1) * VARIABLES + variables1.var].c) = COST

Example

minimum_weight_alldifferent

$$\left(\begin{array}{l} \left\{ \begin{array}{l} \text{var} - 2, \\ \text{var} - 3, \\ \text{var} - 1, \\ \text{var} - 4 \end{array} \right\}, \\ \left. \begin{array}{l} i - 1 \quad j - 1 \quad c - 4, \\ i - 1 \quad j - 2 \quad c - 1, \\ i - 1 \quad j - 3 \quad c - 7, \\ i - 1 \quad j - 4 \quad c - 0, \\ i - 2 \quad j - 1 \quad c - 1, \\ i - 2 \quad j - 2 \quad c - 0, \\ i - 2 \quad j - 3 \quad c - 8, \\ i - 2 \quad j - 4 \quad c - 2, \\ i - 3 \quad j - 1 \quad c - 3, \\ i - 3 \quad j - 2 \quad c - 2, \\ i - 3 \quad j - 3 \quad c - 1, \\ i - 3 \quad j - 4 \quad c - 6, \\ i - 4 \quad j - 1 \quad c - 0, \\ i - 4 \quad j - 2 \quad c - 0, \\ i - 4 \quad j - 3 \quad c - 6, \\ i - 4 \quad j - 4 \quad c - 5 \end{array} \right\}, 17 \end{array} \right)$$

The cost 17 corresponds to the sum $\text{MATRIX}[(1 - 1) \cdot 4 + 2].c + \text{MATRIX}[(2 -$

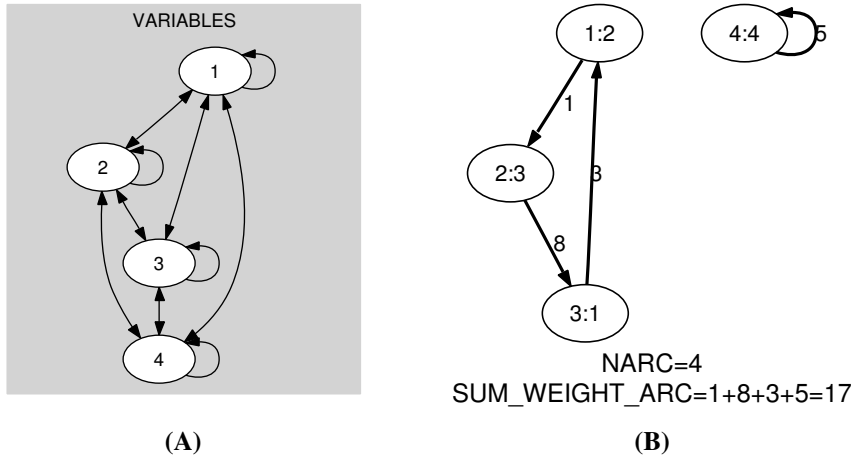


Figure 4.310: Initial and final graph of the minimum_weight_alldifferent constraint

$1) \cdot 4 + 3].c + \text{MATRIX}[(3 - 1) \cdot 4 + 1].c + \text{MATRIX}[(4 - 1) \cdot 4 + 4].c = \text{MATRIX}[2].c + \text{MATRIX}[7].c + \text{MATRIX}[9].c + \text{MATRIX}[16].c = 1 + 8 + 3 + 5.$ Parts (A) and (B) of Figure 4.310 respectively show the initial and final graph. Since we use the **SUM_WEIGHT_ARC** graph property, the arcs of the final graph are stressed in bold; We also indicate their corresponding weight.

Graph model

Since each variable takes one value, and because of the arc constraint `variables1 = variables.key`, each vertex of the initial graph belongs to the final graph and has exactly

one successor. Therefore the sum of the out-degrees of the vertices of the final graph is equal to the number of vertices of the final graph. Since the sum of the in-degrees is equal to the sum of the out-degrees, it is also equal to the number of vertices of the final graph. Since `NTREE` = 0, each vertex of the final graph belongs to a circuit. Therefore each vertex of the final graph has at least one predecessor. Since we saw that the sum of the in-degrees is equal to the number of vertices of the final graph, each vertex of the final graph has exactly one predecessor. We conclude that the final graph consists of a set of vertex-disjoint elementary circuits.

Finally the graph constraint expresses the fact that the `COST` variable is equal to the sum of the elementary costs associated to each variable-value assignment. All these elementary costs are recorded in the `MATRIX` collection. More precisely, the cost c_{ij} is recorded in the attribute `c` of the $((i - 1) \cdot |\text{VARIABLES}| + j)^{th}$ entry of the `MATRIX` collection. This is ensured by the `increasing` restriction which enforces the fact that the items of the `MATRIX` collection are sorted in lexicographically increasing order according to attributes `i` and `j`.

Algorithm

A filtering algorithm is described in [132]. It can be used for handling both side of the `minimum_weight_alldifferent` constraint:

- Evaluating a lower bound of the `COST` variable and pruning the variables of the `VARIABLES` collection in order to not exceed the maximum value of `COST`.
- Evaluating an upper bound of the `COST` variable and pruning the variables of the `VARIABLES` collection in order to not be under the minimum value of `COST`.

See also

`alldifferent`, `global_cardinality_with_costs`, `weighted_partial_alldiff`.

Key words

cost filtering constraint, assignment, cost matrix, weighted assignment, `one_succ`.

4.143 nclass

Origin	Derived from <i>nvalue</i> .
Constraint	<code>nclass(NCLASS, VARIABLES, PARTITIONS)</code>
Type(s)	<code>VALUES : collection(val - int)</code>
Argument(s)	<code>NCLASS : dvar</code> <code>VARIABLES : collection(var - dvar)</code> <code>PARTITIONS : collection(p - VALUES)</code>
Restriction(s)	<code>required(VALUES, val)</code> <code>distinct(VALUES, val)</code> <code>NCLASS ≥ 0</code> <code>NCLASS ≤ min(VARIABLES , PARTITIONS)</code> <code>required(VARIABLES, var)</code> <code>required(PARTITIONS, p)</code> <code> PARTITIONS ≥ 2</code>
Purpose	Number of partitions of the collection <code>PARTITIONS</code> such that at least one value is assigned to at least one variable of the collection <code>VARIABLES</code> .
Arc input(s)	<code>VARIABLES</code>
Arc generator	<code>CLIQUE ↦ collection(variables1, variables2)</code>
Arc arity	2
Arc constraint(s)	<code>in_same_partition(variables1.var, variables2.var, PARTITIONS)</code>
Graph property(ies)	<code>NSCC = NCLASS</code>

Example	$\text{nclass} \left(\begin{array}{c} \left(\begin{array}{c} \text{var} - 3, \\ \text{var} - 2, \\ \text{var} - 7, \\ \text{var} - 2, \\ \text{var} - 6 \end{array} \right), \\ 2, \left(\begin{array}{c} \text{p} - \{\text{val} - 1, \text{val} - 3\}, \\ \text{p} - \{\text{val} - 4\}, \\ \text{p} - \{\text{val} - 2, \text{val} - 6\} \end{array} \right) \end{array} \right)$
----------------	---

Parts (A) and (B) of Figure 4.311 respectively show the initial and final graph. Since we use the **NSCC** graph property we show the different strongly connected components of the final graph. Each strongly connected component corresponds to one class of values which were assigned to some variables of the `VARIABLES` collection. We effectively use two classes of values that respectively correspond to values `{3}` and `{2, 6}`. Note that we do not consider value 7 since it does not belong to the different classes of values we gave: all corresponding arc constraints do not hold.

Algorithm [33, 106].

See also `nvalue`, `nequivalence`, `ninterval`, `npair`, `in_same_partition`.

Key words counting constraint, value partitioning constraint, number of distinct equivalence classes, partition, strongly connected component, equivalence.

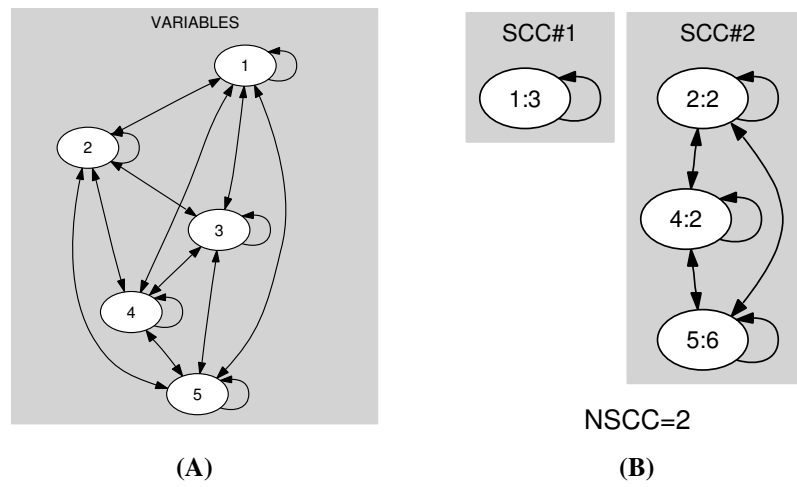


Figure 4.311: Initial and final graph of the nclass constraint

20000128

673

4.144 nequivalence

Origin	Derived from <code>nvalue</code> .
Constraint	<code>nequivalence(NEQUIV, M, VARIABLES)</code>
Argument(s)	<code>NEQUIV</code> : dvar <code>M</code> : int <code>VARIABLES</code> : collection(var – dvar)
Restriction(s)	$NEQUIV \geq \min(1, VARIABLES)$ $NEQUIV \leq \min(M, VARIABLES)$ $M > 0$ <code>required(VARIABLES, var)</code>
Purpose	<code>NEQUIV</code> is the number of distinct rests obtained by dividing the variables of the collection <code>VARIABLES</code> by <code>M</code> .
Arc input(s)	<code>VARIABLES</code>
Arc generator	<code>CLIQUE</code> \mapsto <code>collection(variables1, variables2)</code>
Arc arity	2
Arc constraint(s)	<code>variables1.var mod M = variables2.var mod M</code>
Graph property(ies)	<code>NSCC = NEQUIV</code>
Example	$\text{nequivalence} \left(2, 3, \left\{ \begin{array}{l} \text{var} - 3, \\ \text{var} - 2, \\ \text{var} - 5, \\ \text{var} - 6, \\ \text{var} - 15, \\ \text{var} - 3, \\ \text{var} - 3 \end{array} \right\} \right)$ <p>Parts (A) and (B) of Figure 4.312 respectively show the initial and final graph. Since we use the <code>NSCC</code> graph property we show the different strongly connected components of the final graph. Each strongly connected component corresponds to one equivalence class: We have two equivalence classes that respectively correspond to values $\{3, 6, 15\}$ and $\{2, 5\}$.</p>
Algorithm	Since constraints $X = Y$ and $X \equiv Y \pmod{M}$ are similar, one should also use a similar algorithm as the one [33, 106] provided for constraint <code>nvalue</code> .
See also	<code>nvalue</code> , <code>nclass</code> , <code>ninterval</code> , <code>npair</code> .
Key words	counting constraint, value partitioning constraint, number of distinct equivalence classes, strongly connected component, equivalence.

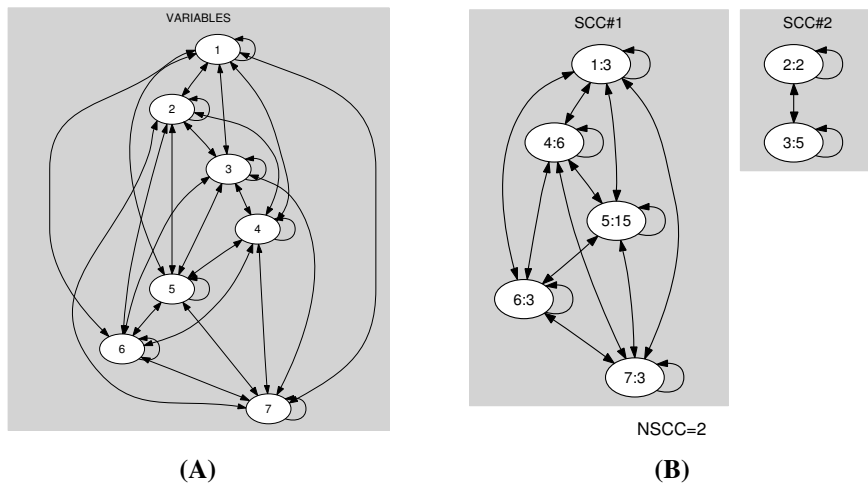


Figure 4.312: Initial and final graph of the nequivalence constraint

4.145 next_element

Origin	N. Beldiceanu
Constraint	<code>next_element(THRESHOLD, INDEX, TABLE, VAL)</code>
Argument(s)	<code>THRESHOLD</code> : dvar <code>INDEX</code> : dvar <code>TABLE</code> : <code>collection(index – int, value – dvar)</code> <code>VAL</code> : dvar
Restriction(s)	<code>INDEX</code> ≥ 1 <code>INDEX</code> \leq TABLE <code>required(TABLE, [index, value])</code> <code>TABLE.index</code> ≥ 1 <code>TABLE.index</code> \leq TABLE <code>distinct(TABLE, index)</code>
Purpose	<code>INDEX</code> is the smallest entry of <code>TABLE</code> strictly greater than <code>THRESHOLD</code> containing value <code>VAL</code> .
Derived Collection(s)	<code>col</code> $\left(\begin{array}{l} \text{ITEM} - \text{collection}(\text{index} - \text{dvar}, \text{value} - \text{dvar}), \\ [\text{item}(\text{index} - \text{THRESHOLD}, \text{value} - \text{VAL})] \end{array} \right)$
Arc input(s)	ITEM TABLE
Arc generator	<i>PRODUCT</i> \mapsto <code>collection(item, table)</code>
Arc arity	2
Arc constraint(s)	<ul style="list-style-type: none"> • <code>item.index</code> $<$ <code>table.index</code> • <code>item.value</code> = <code>table.value</code>
Graph property(ies)	$\overline{\text{NARC}} > 0$
Sets	$\text{SUCC} \mapsto$ $\left[\begin{array}{l} \text{source}, \\ \text{variables} - \text{col}(\text{VARIABLES} - \text{collection}(\text{var} - \text{dvar}), [\text{item}(\text{var} - \text{TABLE.index})]) \end{array} \right]$
Constraint(s) on sets	<code>minimum(INDEX, variables)</code>
Example	$\text{next_element} \left(2, 3, \left\{ \begin{array}{l} \text{index} - 1 \quad \text{value} - 1, \\ \text{index} - 2 \quad \text{value} - 8, \\ \text{index} - 3 \quad \text{value} - 9, \\ \text{index} - 4 \quad \text{value} - 5, \\ \text{index} - 5 \quad \text{value} - 9 \end{array} \right\}, 9 \right)$

The `next_element` constraint holds since 3 is the smallest entry located after entry 2 that contains value 9. Parts (A) and (B) of Figure 4.313 respectively show the initial and final graph associated to the second graph constraint. Since we use the $\overline{\text{NARC}}$ graph property, the arcs of the final graph are stressed in bold.

Automaton

Figure 4.314 depicts the automaton associated to the `next_element` constraint. Let I_k and V_k respectively be the `index` and the `value` attributes of the k^{th} item of the `TABLE` collections. To each quintuple $(\text{THRESHOLD}, \text{INDEX}, \text{VAL}, I_k, V_k)$ corresponds a signature variable S_k as well as the following signature constraint:

$$\begin{aligned} ((I_k \leq \text{THRESHOLD}) \wedge (I_k < \text{INDEX}) \wedge (V_k = \text{VAL})) &\Leftrightarrow S_k = 0 \wedge \\ ((I_k \leq \text{THRESHOLD}) \wedge (I_k < \text{INDEX}) \wedge (V_k \neq \text{VAL})) &\Leftrightarrow S_k = 1 \wedge \\ ((I_k \leq \text{THRESHOLD}) \wedge (I_k = \text{INDEX}) \wedge (V_k = \text{VAL})) &\Leftrightarrow S_k = 2 \wedge \\ ((I_k \leq \text{THRESHOLD}) \wedge (I_k = \text{INDEX}) \wedge (V_k \neq \text{VAL})) &\Leftrightarrow S_k = 3 \wedge \\ ((I_k \leq \text{THRESHOLD}) \wedge (I_k > \text{INDEX}) \wedge (V_k = \text{VAL})) &\Leftrightarrow S_k = 4 \wedge \\ ((I_k \leq \text{THRESHOLD}) \wedge (I_k > \text{INDEX}) \wedge (V_k \neq \text{VAL})) &\Leftrightarrow S_k = 5 \wedge \\ ((I_k > \text{THRESHOLD}) \wedge (I_k < \text{INDEX}) \wedge (V_k = \text{VAL})) &\Leftrightarrow S_k = 6 \wedge \\ ((I_k > \text{THRESHOLD}) \wedge (I_k < \text{INDEX}) \wedge (V_k \neq \text{VAL})) &\Leftrightarrow S_k = 7 \wedge \\ ((I_k > \text{THRESHOLD}) \wedge (I_k = \text{INDEX}) \wedge (V_k = \text{VAL})) &\Leftrightarrow S_k = 8 \wedge \\ ((I_k > \text{THRESHOLD}) \wedge (I_k = \text{INDEX}) \wedge (V_k \neq \text{VAL})) &\Leftrightarrow S_k = 9 \wedge \\ ((I_k > \text{THRESHOLD}) \wedge (I_k > \text{INDEX}) \wedge (V_k = \text{VAL})) &\Leftrightarrow S_k = 10 \wedge \\ ((I_k > \text{THRESHOLD}) \wedge (I_k > \text{INDEX}) \wedge (V_k \neq \text{VAL})) &\Leftrightarrow S_k = 11. \end{aligned}$$

The automaton is constructed in order to fulfill the following conditions:

- We look for an item of the `TABLE` collection such that $\text{INDEX}_i > \text{THRESHOLD}$ and $\text{INDEX}_i = \text{INDEX}$ and $\text{VALUE}_i = \text{VAL}$,
- There should not exist any item of the `TABLE` collection such that $\text{INDEX}_i > \text{THRESHOLD}$ and $\text{INDEX}_i < \text{INDEX}$ and $\text{VALUE}_i = \text{VAL}$.

Usage

Originally introduced for modelling the fact that a nucleotide has to be consumed as soon as possible at cycle `INDEX` after a given cycle represented by variable `THRESHOLD`.

See also

`minimum_greater_than`, `next_greater_element`.

Key words

data constraint, minimum, table, automaton, automaton without counters, centered cyclic(3) constraint network(1), derived collection.

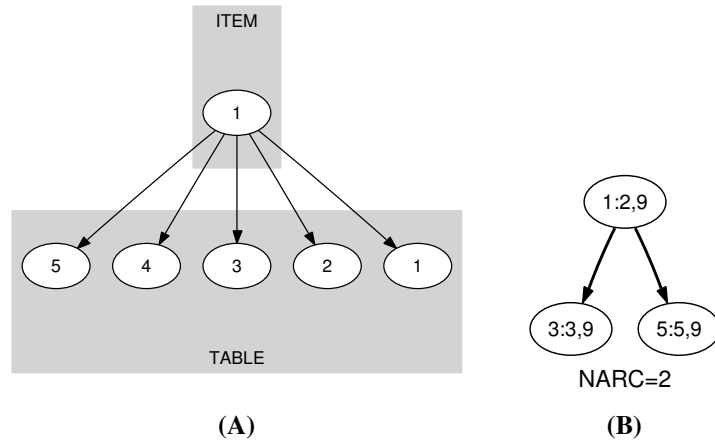


Figure 4.313: Initial and final graph of the next_element constraint

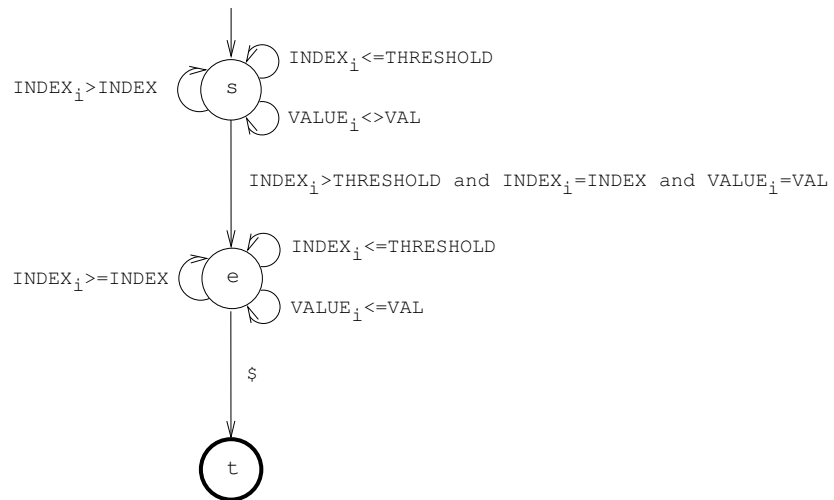


Figure 4.314: Automaton of the next_element constraint

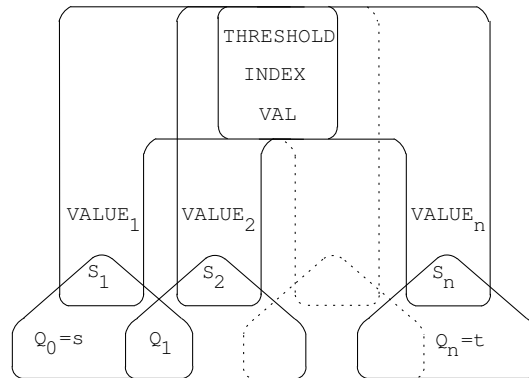


Figure 4.315: Hypergraph of the reformulation corresponding to the automaton of the `next_element` constraint

4.146 next_greater_element

Origin	M. Carlsson
Constraint	<code>next_greater_element(VAR1, VAR2, VARIABLES)</code>
Argument(s)	<code>VAR1</code> : dvar <code>VAR2</code> : dvar <code>VARIABLES</code> : <code>collection(var - dvar)</code>
Restriction(s)	$ \text{VARIABLES} > 0$ <code>required(VARIABLES, var)</code>
Purpose	<p>VAR2 is the value strictly greater than VAR1 located at the smallest possible entry of the table TABLE. In addition, the variables of the collection VARIABLES are sorted in strictly increasing order.</p>
Derived Collection(s)	<code>col(V - collection(var - dvar), [item(var - VAR1)])</code>
Arc input(s)	VARIABLES
Arc generator	$\text{PATH} \mapsto \text{collection}(\text{variables1}, \text{variables2})$
Arc arity	2
Arc constraint(s)	<code>variables1.var < variables2.var</code>
Graph property(ies)	$\text{NARC} = \text{VARIABLES} - 1$
Arc input(s)	V VARIABLES
Arc generator	$\text{PRODUCT} \mapsto \text{collection}(v, \text{variables})$
Arc arity	2
Arc constraint(s)	<code>v.var < variables.var</code>
Graph property(ies)	$\text{NARC} > 0$
Sets	$\text{SUCC} \mapsto [\text{source}, \text{variables}]$
Constraint(s) on sets	<code>minimum(VAR2, variables)</code>

Example `next_greater_element` $\left(7, 8, \left\{ \begin{array}{l} \text{var} - 3, \\ \text{var} - 5, \\ \text{var} - 8, \\ \text{var} - 9 \end{array} \right\} \right)$

The `next_greater_element` constraint holds since:

- VAR2 is fixed to the first value 8 strictly greater than VAR1 = 7,
- The var attributes of the items of the collection VARIABLES are sorted in strictly increasing order.

Parts (A) and (B) of Figure 4.316 respectively show the initial and final graph associated to the second graph constraint. Since we use the **NARC** graph property, the arcs of the final graph are stressed in bold.

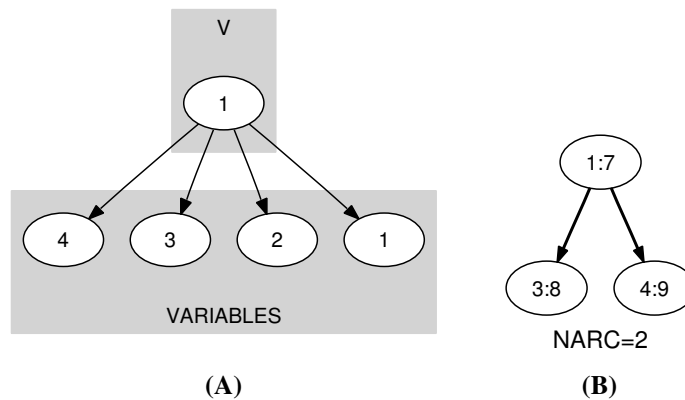


Figure 4.316: Initial and final graph of the `next_greater_element` constraint

Signature

Since the first graph constraint uses the *PATH* arc generator on the `VARIABLES` collection, the number of arcs of the corresponding initial graph is equal to $|\text{VARIABLES}| - 1$. Therefore the maximum number of arcs of the final graph is equal to $|\text{VARIABLES}| - 1$. For this reason we can rewrite $\text{NARC} = |\text{VARIABLES}| - 1$ to $\text{NARC} \geq |\text{VARIABLES}| - 1$ and simplify **NARC** to **NARC**.

Usage

Originally introduced for modelling the fact that a nucleotide has to be consumed as soon as possible at cycle VAR2 after a given cycle VAR1.

Remark

Similar to the `minimum_greater_than` constraint, except for the fact that the `var` attributes are sorted.

See also

`minimum_greater_than`, `next_element`.

Key words

order constraint, minimum, data constraint, table, derived collection.

4.147 ninterval

Origin	Derived from <code>nvalue</code> .
Constraint	<code>ninterval(NVAL, VARIABLES, SIZE_INTERVAL)</code>
Argument(s)	<code>NVAL</code> : <code>dvar</code> <code>VARIABLES</code> : <code>collection(var - dvar)</code> <code>SIZE_INTERVAL</code> : <code>int</code>
Restriction(s)	$NVAL \geq \min(1, VARIABLES)$ $NVAL \leq VARIABLES $ <code>required(VARIABLES, var)</code> $SIZE_INTERVAL > 0$
Purpose	Consider the intervals of the form $[SIZE_INTERVAL \cdot k, SIZE_INTERVAL \cdot k + SIZE_INTERVAL - 1]$ where k is an integer. <code>NVAL</code> is the number of intervals for which at least one value is assigned to at least one variable of the collection <code>VARIABLES</code> .
Arc input(s)	<code>VARIABLES</code>
Arc generator	<code>CLIQUE</code> \mapsto <code>collection(variables1, variables2)</code>
Arc arity	2
Arc constraint(s)	<code>variables1.var/SIZE_INTERVAL = variables2.var/SIZE_INTERVAL</code>
Graph property(ies)	<code>NSCC = NVAL</code>
Example	$\text{ninterval} \left(2, \left\{ \begin{array}{l} \text{var} - 3, \\ \text{var} - 1, \\ \text{var} - 9, \\ \text{var} - 1, \\ \text{var} - 9 \end{array} \right\}, 4 \right)$ <p>Parts (A) and (B) of Figure 4.317 respectively show the initial and final graph. Since we use the <code>NSCC</code> graph property we show the different strongly connected components of the final graph. Each strongly connected component corresponds to those values of an interval which are assigned to some variables of the <code>VARIABLES</code> collection. The values 1, 3 and the value 9 which respectively correspond to intervals $[0, 3]$ and $[7, 9]$ are assigned to the variables of the <code>VARIABLES</code> collection.</p>
Usage	The <code>ninterval</code> constraint is useful for counting the number of effectively used periods, no matter how many time each period is used. A period can for example stand for a hour or for a day.
Algorithm	[33, 106].
See also	<code>nvalue</code> , <code>nclass</code> , <code>nequivalence</code> , <code>npair</code> .
Key words	counting constraint, value partitioning constraint, number of distinct equivalence classes, interval, strongly connected component, equivalence.

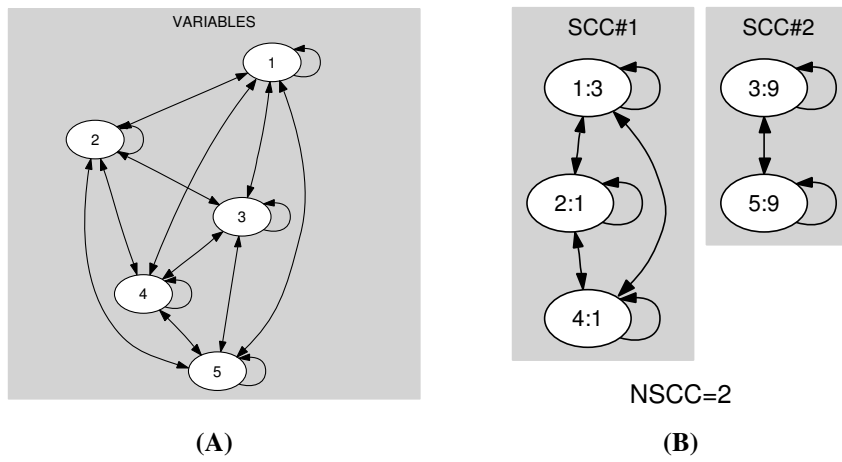


Figure 4.317: Initial and final graph of the ninterval constraint

4.148 no_peak

Origin	Derived from peak.
Constraint	no_peak(VARIABLES)
Argument(s)	VARIABLES : collection(var - dvar)
Restriction(s)	VARIABLES > 0 required(VARIABLES, var)

Purpose

A variable V_k ($1 < k < m$) of the sequence of variables $VARIABLES = V_1, \dots, V_m$ is a *peak* if and only if there exist an i ($1 < i \leq k$) such that $V_{i-1} < V_i$ and $V_i = V_{i+1} = \dots = V_k$ and $V_k > V_{k+1}$. The total number of peaks of the sequence of variables $VARIABLES$ is equal to 0.

Example

$$\text{no_peak} \left(\left(\begin{array}{l} \text{var} - 1, \\ \text{var} - 1, \\ \text{var} - 4, \\ \text{var} - 8, \\ \text{var} - 8 \end{array} \right) \right)$$

The previous constraint holds since the sequence 1 1 4 8 8 does not contain any peak.

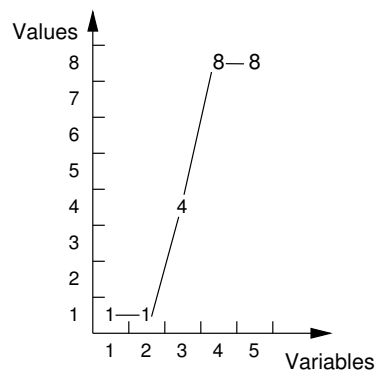


Figure 4.318: A sequence without any peak

Automaton

Figure 4.319 depicts the automaton associated to the no_peak constraint. To each pair of consecutive variables (VAR_i, VAR_{i+1}) of the collection $VARIABLES$ corresponds a signature variable S_i . The following signature constraint links VAR_i, VAR_{i+1} and S_i : $(VAR_i < VAR_{i+1} \Leftrightarrow S_i = 0) \wedge (VAR_i = VAR_{i+1} \Leftrightarrow S_i = 1) \wedge (VAR_i > VAR_{i+1} \Leftrightarrow S_i = 2)$.

See also

peak, no_valley, valley.

Key words

sequence, automaton, automaton without counters, sliding cyclic(1) constraint network(1).

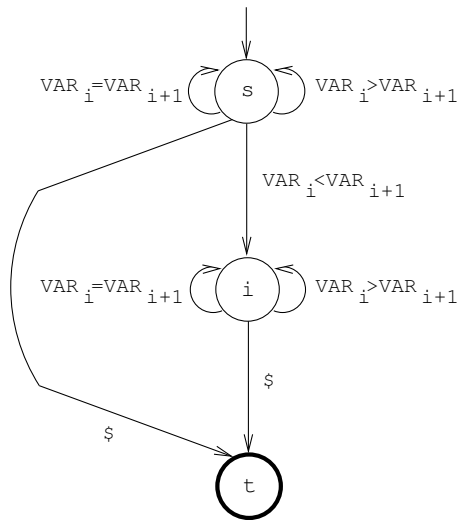


Figure 4.319: Automaton of the no_peak constraint

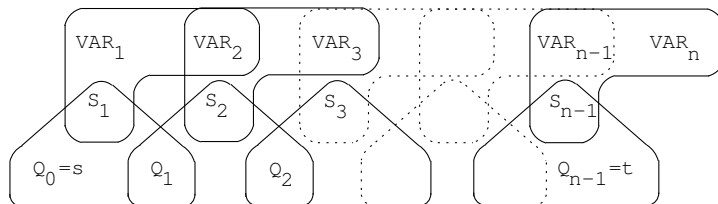


Figure 4.320: Hypergraph of the reformulation corresponding to the automaton of the no_peak constraint

4.149 no_valley

Origin Derived from valley.

Constraint no_valley(VARIABLES)

Argument(s) VARIABLES : collection(var – dvar)

Restriction(s) |VARIABLES| > 0
required(VARIABLES, var)

Purpose

A variable V_k ($1 < k < m$) of the sequence of variables $VARIABLES = V_1, \dots, V_m$ is a *valley* if and only if there exist an i ($1 < i \leq k$) such that $V_{i-1} > V_i$ and $V_i = V_{i+1} = \dots = V_k$ and $V_k < V_{k+1}$. The total number of valleys of the sequence of variables $VARIABLES$ is equal to 0.

Example

no_valley $\left(\left(\begin{array}{l} \text{var} - 1, \\ \text{var} - 1, \\ \text{var} - 4, \\ \text{var} - 8, \\ \text{var} - 8, \\ \text{var} - 2 \end{array} \right) \right)$

The previous constraint holds since the sequence 1 1 4 8 8 2 does not contain any valley.

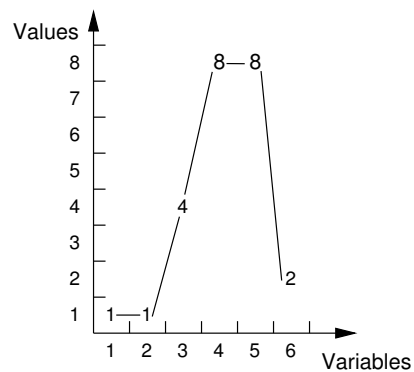


Figure 4.321: A sequence without any valley

Automaton

Figure 4.322 depicts the automaton associated to the no_valley constraint. To each pair of consecutive variables (VAR_i, VAR_{i+1}) of the collection VARIABLES corresponds a signature variable S_i . The following signature constraint links VAR_i , VAR_{i+1} and S_i : $(VAR_i < VAR_{i+1} \Leftrightarrow S_i = 0) \wedge (VAR_i = VAR_{i+1} \Leftrightarrow S_i = 1) \wedge (VAR_i > VAR_{i+1} \Leftrightarrow S_i = 2)$.

See also

valley, no_peak, peak.

Key words

sequence, automaton, automaton without counters, sliding cyclic(1) constraint network(1).

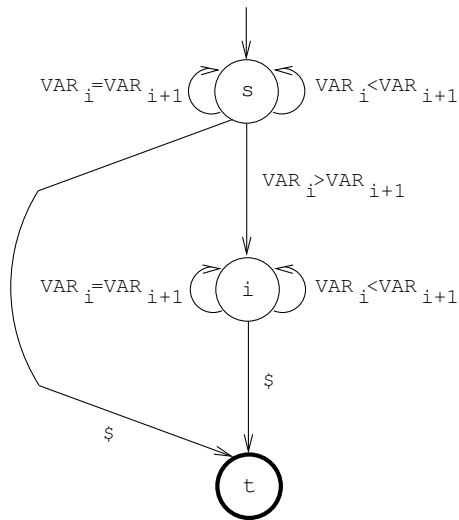


Figure 4.322: Automaton of the no_valley constraint

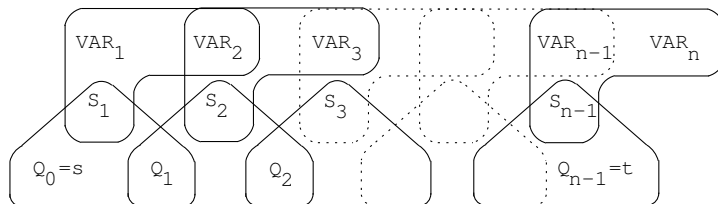


Figure 4.323: Hypergraph of the reformulation corresponding to the automaton of the no_valley constraint

4.150 not_all_equal

Origin	CHIP
Constraint	<code>not_all_equal(VARIABLES)</code>
Argument(s)	<code>VARIABLES : collection(var - dvar)</code>
Restriction(s)	<code>required(VARIABLES, var)</code> <code> VARIABLES > 1</code>
Purpose	The variables of the collection <code>VARIABLES</code> should take more than one single value.
<hr/>	
Arc input(s)	<code>VARIABLES</code>
Arc generator	<code>CLIQUE</code> \mapsto <code>collection(variables1, variables2)</code>
Arc arity	2
Arc constraint(s)	<code>variables1.var = variables2.var</code>
Graph property(ies)	<code>NSCC > 1</code>
<hr/>	
Example	$\text{not_all_equal} \left(\left(\begin{array}{c} \text{var} - 3, \\ \text{var} - 1, \\ \text{var} - 3, \\ \text{var} - 3, \\ \text{var} - 3 \end{array} \right) \right)$
	Parts (A) and (B) of Figure 4.324 respectively show the initial and final graph. Since we use the <code>NSCC</code> graph property we show the different strongly connected components of the final graph. Each strongly connected component corresponds to one value which is assigned to some variables of the <code>VARIABLES</code> collection. The <code>not_all_equal</code> holds since the final graph contains more than one strongly connected component.
Automaton	Figure 4.325 depicts the automaton associated to the <code>not_all_equal</code> constraint. To each pair of consecutive variables ($\text{VAR}_i, \text{VAR}_{i+1}$) of the collection <code>VARIABLES</code> corresponds a signature variable S_i . The following signature constraint links $\text{VAR}_i, \text{VAR}_{i+1}$ and S_i : $\text{VAR}_i = \text{VAR}_{i+1} \Leftrightarrow S_i$.
Algorithm	If the intersection of the domains of the variables of the <code>VARIABLES</code> collection is empty the <code>not_all_equal</code> constraint is entailed. Otherwise, when only one single variable V remains not fixed, remove the unique value (unique since the constraint is not entailed) taken by the other variables from the domain of V .
See also	<code>nvalue</code> .
Key words	value constraint, disequality, automaton, automaton without counters, sliding cyclic(1) constraint network(1), equivalence.

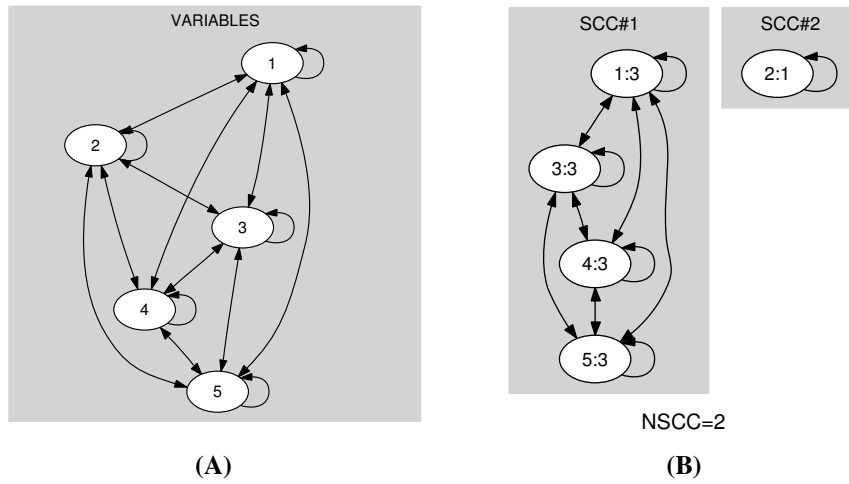


Figure 4.324: Initial and final graph of the not_all_equal constraint

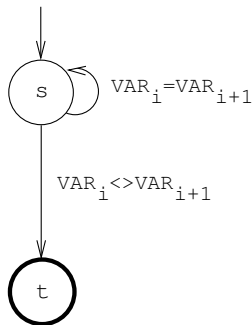


Figure 4.325: Automaton of the not_all_equal constraint

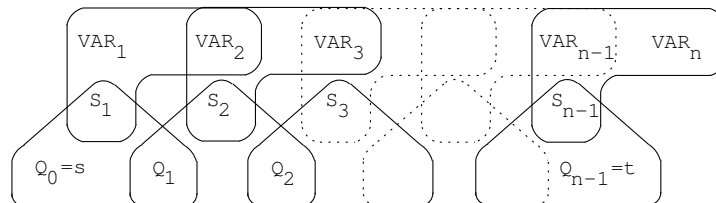


Figure 4.326: Hypergraph of the reformulation corresponding to the automaton of the not_all_equal constraint

4.151 not_in

Origin	Derived from in.
Constraint	<code>not_in(VAR, VALUES)</code>
Argument(s)	VAR : dvar VALUES : collection(val – int)
Restriction(s)	required(VALUES, val) distinct(VALUES, val)
Purpose	Remove the values of the VALUES collection from domain variable VAR.
Derived Collection(s)	$\text{col}(\text{VARIABLES} - \text{collection}(\text{var} - \text{dvar}), [\text{item}(\text{var} - \text{VAR})])$
Arc input(s)	VARIABLES VALUES
Arc generator	$\text{PRODUCT} \mapsto \text{collection}(\text{variables}, \text{values})$
Arc arity	2
Arc constraint(s)	<code>variables.var = values.val</code>
Graph property(ies)	$\text{NARC} = 0$
Example	<code>not_in(2, {val – 1, val – 3})</code>

Figure 4.327 shows the initial graph associated to the previous example. Since we use the $\text{NARC} = 0$ graph property the final graph is empty.

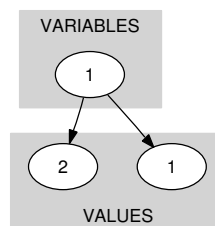
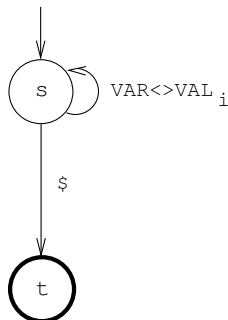


Figure 4.327: Initial graph of the `not_in` constraint (the final graph is empty)

Signature Since 0 is the smallest number of arcs of the final graph we can rewrite $\text{NARC} = 0$ to $\text{NARC} \leq 0$. This leads to simplify NARC to NARC.

Automaton	Figure 4.328 depicts the automaton associated to the <code>not_in</code> constraint. Let VAL_i be the <code>val</code> attribute of the i^{th} item of the <code>VALUES</code> collection. To each pair (VAR, VAL_i) corresponds a 0-1 signature variable S_i as well as the following signature constraint: $VAR = VAL_i \Leftrightarrow S_i$.
Remark	Entailment occurs immediately after posting this constraint.
Used in	<code>group</code> .
See also	<code>in</code> .
Key words	value constraint, unary constraint, excluded, disequality, domain definition, automaton, automaton without counters, centered cyclic(1) constraint network(1), derived collection.

Figure 4.328: Automaton of the `not_in` constraint

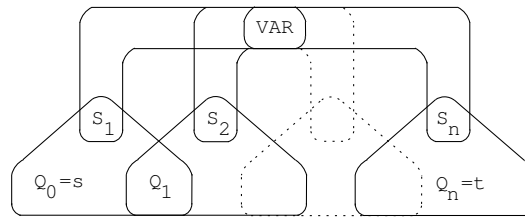


Figure 4.329: Hypergraph of the reformulation corresponding to the automaton of the `not_in` constraint

20030820

693

4.152 npair

Origin	Derived from <i>nvalue</i> .
Constraint	<code>npair(NVAL, PAIRS)</code>
Argument(s)	<p><code>NVAL</code> : <code>dvar</code></p> <p><code>PAIRS</code> : <code>collection(x - dvar, y - dvar)</code></p>
Restriction(s)	<p>$NVAL \geq \min(1, PAIRS)$</p> <p>$NVAL \leq PAIRS$</p> <p><code>required(PAIRS, [x, y])</code></p>
Purpose	<p><code>NVAL</code> is the number of distinct pairs of values assigned to the pairs of variables of the collection <code>PAIRS</code>.</p>
<hr/>	
Arc input(s)	<code>PAIRS</code>
Arc generator	<code>CLIQUE</code> \mapsto <code>collection(pairs1, pairs2)</code>
Arc arity	2
Arc constraint(s)	<ul style="list-style-type: none"> • <code>pairs1.x = pairs2.x</code> • <code>pairs1.y = pairs2.y</code>
Graph property(ies)	<code>NSCC = NVAL</code>
<hr/>	
Example	$\text{npair} \left(2, \left\{ \begin{array}{l} x - 3 \quad y - 1, \\ x - 1 \quad y - 5, \\ x - 3 \quad y - 1, \\ x - 3 \quad y - 1, \\ x - 1 \quad y - 5 \end{array} \right\} \right)$ <p>Parts (A) and (B) of Figure 4.330 respectively show the initial and final graph. Since we use the NSCC graph property we show the different strongly connected components of the final graph. Each strongly connected component corresponds to one pair of values which is assigned to some pairs of variables of the <code>PAIRS</code> collection. In our example we have the following pairs of values: (3,1) and (1,5).</p>
Remark	This is an example of a <i>number of distinct values</i> constraint where there is more than one attribute that is associated to each vertex of the final graph.
See also	<code>nvalue</code> , <code>nclass</code> , <code>nequivalence</code> , <code>ninterval</code> .
Key words	counting constraint, value partitioning constraint, number of distinct equivalence classes, pair, strongly connected component, equivalence.

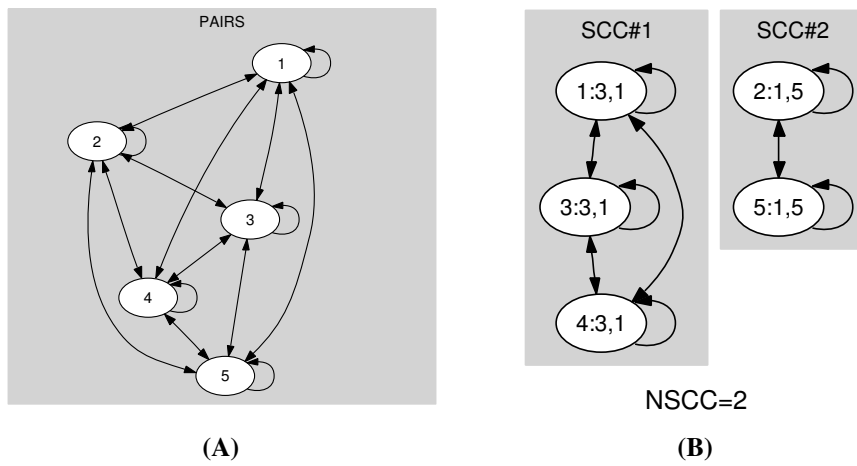


Figure 4.330: Initial and final graph of the npair constraint

4.153 nset_of_consecutive_values

Origin	N. Beldiceanu
Constraint	<code>nset_of_consecutive_values(N, VARIABLES)</code>
Argument(s)	<code>N</code> : dvar <code>VARIABLES</code> : collection(var – dvar)
Restriction(s)	$N \geq 1$ $N \leq \text{VARIABLES} $ <code>required(VARIABLES, var)</code>
Purpose	<code>N</code> is the number of set of consecutive values used by the variables of the collection <code>VARIABLES</code> .
Arc input(s)	<code>VARIABLES</code>
Arc generator	<code>CLIQUE</code> \mapsto <code>collection(variables1, variables2)</code>
Arc arity	2
Arc constraint(s)	<code>abs(variables1.var – variables2.var) \leq 1</code>
Graph property(ies)	<code>NSCC = N</code>
Example	$\text{nset_of_consecutive_values} \left(2, \left\{ \begin{array}{l} \text{var} - 3, \\ \text{var} - 1, \\ \text{var} - 7, \\ \text{var} - 1, \\ \text{var} - 1, \\ \text{var} - 2, \\ \text{var} - 8 \end{array} \right\} \right)$ <p>In this example, the variables of the collection <code>VARIABLES</code> use the following two sets of consecutive values: $\{1, 2, 3\}$ and $\{7, 8\}$. Parts (A) and (B) of Figure 4.331 respectively show the initial and final graph. Since we use the <code>NSCC</code> graph property, we show the two strongly connected components of the final graph.</p>
Graph model	Since the arc constraint is symmetric each strongly connected component of the final graph corresponds exactly to one connected component of the final graph.
Usage	Used for specifying the fact that the values have to be used in a compact way is achieved by setting <code>N</code> to 1.
See also	<code>min_size_set_of_consecutive_var</code> .
Key words	value constraint, consecutive values, strongly connected component.

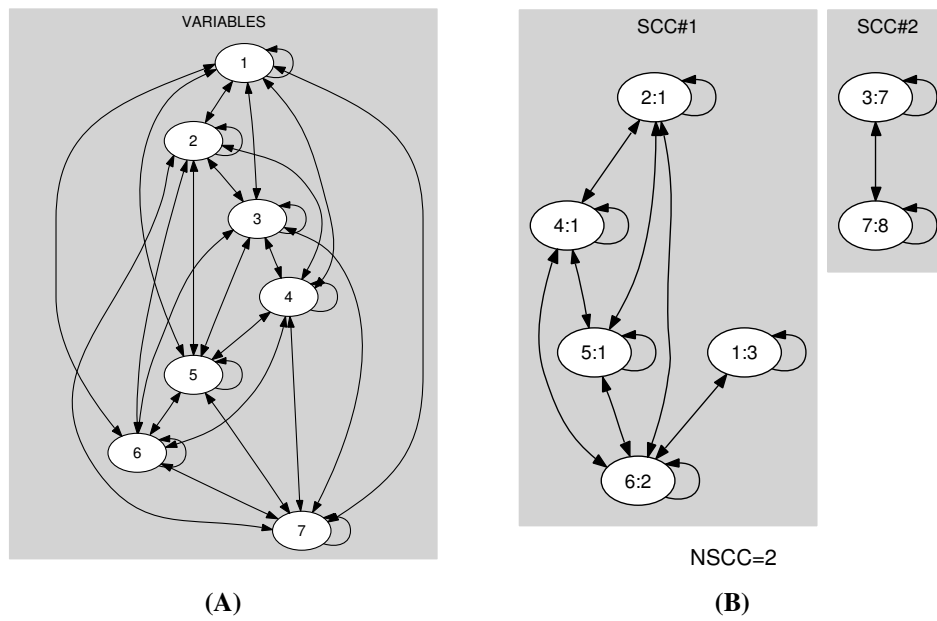


Figure 4.331: Initial and final graph of the `nset_of_consecutive_values` constraint

4.154 nvalue

Origin	[73]
Constraint	<code>nvalue(NVAL, VARIABLES)</code>
Synonym(s)	<code>cardinality_on_attributes_values.</code>
Argument(s)	<code>NVAL</code> : <code>dvar</code> <code>VARIABLES</code> : <code>collection(var – dvar)</code>
Restriction(s)	$NVAL \geq \min(1, VARIABLES)$ $NVAL \leq VARIABLES $ <code>required(VARIABLES, var)</code>
Purpose	NVAL is the number of distinct values taken by the variables of the collection VARIABLES.
<hr/>	
Arc input(s)	VARIABLES
Arc generator	<i>CLIQUE</i> \mapsto <code>collection(variables1, variables2)</code>
Arc arity	2
Arc constraint(s)	<code>variables1.var = variables2.var</code>
Graph property(ies)	<code>NSCC = NVAL</code>
<hr/>	
Example	$nvalue \left(4, \left\{ \begin{array}{l} \text{var} - 3, \\ \text{var} - 1, \\ \text{var} - 7, \\ \text{var} - 1, \\ \text{var} - 6 \end{array} \right\} \right)$ <p>Parts (A) and (B) of Figure 4.332 respectively show the initial and final graph. Since we use the NSCC graph property we show the different strongly connected components of the final graph. Each strongly connected component corresponds to one value which is assigned to some variables of the VARIABLES collection. The 4 following values 1, 3, 6 and 7 are used by the variables of the VARIABLES collection.</p>
Automaton	Figure 4.333 depicts the automaton associated to the <code>nvalue</code> constraint. To each item of the collection VARIABLES corresponds a signature variable S_i , which is equal to 0.
Usage	This constraint occurs in many practical applications. In the context of timetabling one wants to set up a limit on the maximum number of activity types it is possible to perform. For frequency allocation problems, one optimisation criteria corresponds to the fact that you want to minimize the number of distinct frequencies that you use all over the entire network. The <code>nvalue</code> constraint generalizes several constraints like:

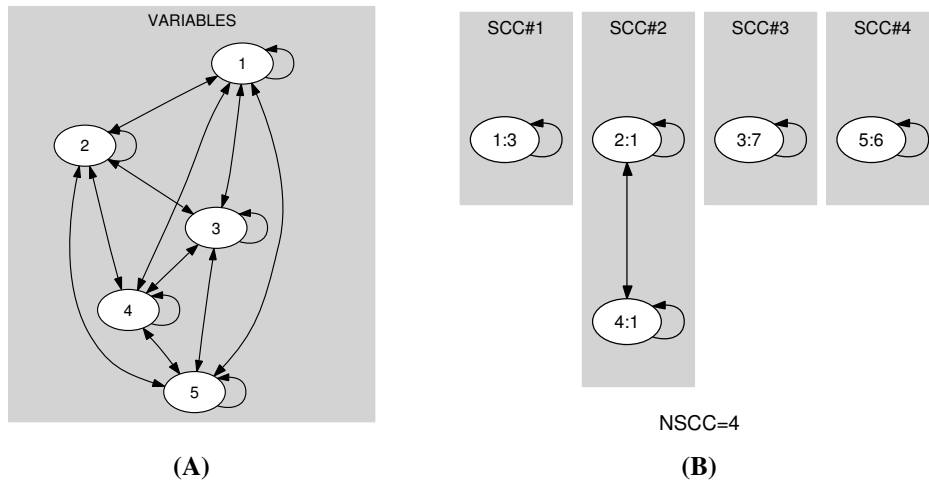


Figure 4.332: Initial and final graph of the nvalue constraint

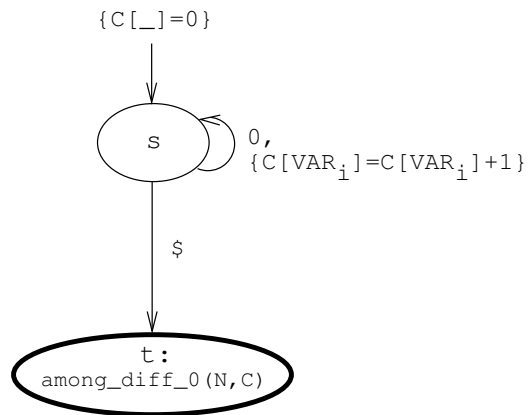


Figure 4.333: Automaton of the nvalue constraint

- `alldifferent(VARIABLES)`: in order to get the `alldifferent` constraint, one has to set `NVAL` to the total number of variables.
- `not_all_equal(VARIABLES)`: in order to get the `not_all_equal` constraint, one has to set the minimum value of `NVAL` to 2.

Remark	This constraint appears in [73, page 339] under the name of <i>Cardinality on Attributes Values</i> . A constraint called <code>k - diff</code> enforcing that a set of variables takes at least k distinct values appears in the PhD thesis of J.-C. Régin [133].
Algorithm	[33, 106, 54].
Used in	<code>track</code> .
See also	<code>alldifferent</code> , <code>not_all_equal</code> , <code>nvalues</code> , <code>nvalues_except_0</code> , <code>npair</code> , <code>nvalue_on_intersection</code> , <code>among_diff_0</code> .
Key words	counting constraint, value partitioning constraint, number of distinct equivalence classes, number of distinct values, strongly connected component, domination, automaton, automaton with array of counters, equivalence.

20000128

701

4.155 `nvalue_on_intersection`

Origin	Derived from <code>common</code> and <code>nvalue</code> .
Constraint	<code>nvalue_on_intersection(NVAL, VARIABLES1, VARIABLES2)</code>
Argument(s)	<code>NVAL</code> : <code>dvar</code> <code>VARIABLES1</code> : <code>collection(var - dvar)</code> <code>VARIABLES2</code> : <code>collection(var - dvar)</code>
Restriction(s)	<code>NVAL</code> \geq 0 <code>NVAL</code> \leq <code> VARIABLES1 </code> <code>NVAL</code> \leq <code> VARIABLES2 </code> <code>required(VARIABLES1, var)</code> <code>required(VARIABLES2, var)</code>
Purpose	<code>NVAL</code> is the number of distinct values which both occur in the <code>VARIABLES1</code> and <code>VARIABLES2</code> collections.
Arc input(s)	<code>VARIABLES1</code> <code>VARIABLES2</code>
Arc generator	<code>PRODUCT</code> \mapsto <code>collection(variables1, variables2)</code>
Arc arity	2
Arc constraint(s)	<code>variables1.var = variables2.var</code>
Graph property(ies)	<code>NCC = NVAL</code>
Example	$\text{nvalue_on_intersection} \left(2, \left\{ \begin{array}{l} \text{var} - 1, \\ \text{var} - 9, \\ \text{var} - 1, \\ \text{var} - 5 \end{array} \right\}, \left\{ \begin{array}{l} \text{var} - 2, \\ \text{var} - 1, \\ \text{var} - 9, \\ \text{var} - 9, \\ \text{var} - 6, \\ \text{var} - 9 \end{array} \right\} \right)$ <p>Parts (A) and (B) of Figure 4.334 respectively show the initial and final graph. Since we use the <code>NCC</code> graph property we show the connected components of the final graph. The variable <code>NVAL</code> is equal to this number of connected components. Observe that all the vertices corresponding to the variables that take values 5, 2 or 6 were removed from the final graph since there is no arc for which the associated equality constraint holds.</p>
See also	<code>nvalue</code> , <code>common</code> , <code>alldifferent_on_intersection</code> , <code>same_intersection</code> .
Key words	counting constraint, number of distinct values, connected component, constraint on the intersection.

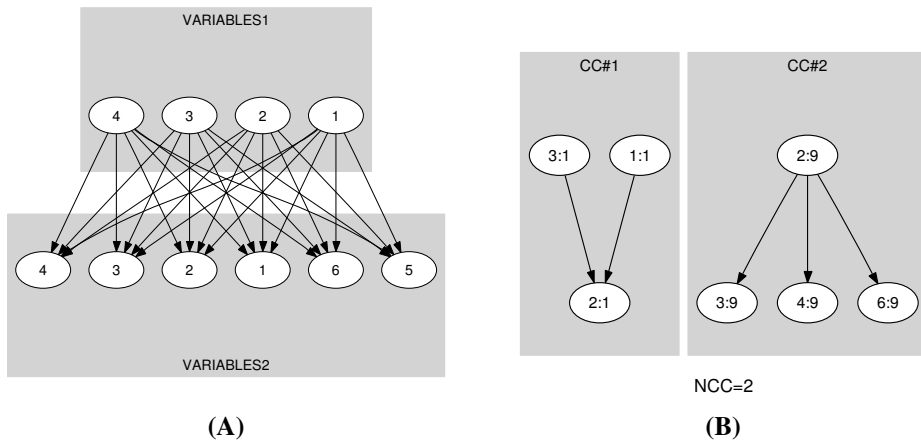


Figure 4.334: Initial and final graph of the `nvalue_on_intersection` constraint

4.156 nvalues

Origin	Inspired by <code>nvalue</code> and <code>count</code> .
Constraint	<code>nvalues(VARIABLES, RELOP, LIMIT)</code>
Argument(s)	<p><code>VARIABLES</code> : <code>collection(var – dvar)</code> <code>RELOP</code> : <code>atom</code> <code>LIMIT</code> : <code>dvar</code></p>
Restriction(s)	<p><code>required(VARIABLES, var)</code> <code>RELOP</code> \in $\{=, \neq, <, \geq, >, \leq\}$</p>
Purpose	<p>Let N be the number of distinct values assigned to the variables of the <code>VARIABLES</code> collection. Enforce condition N <code>RELOP</code> <code>LIMIT</code> to hold.</p>
Arc input(s)	<code>VARIABLES</code>
Arc generator	<code>CLIQUE</code> \mapsto <code>collection(variables1, variables2)</code>
Arc arity	2
Arc constraint(s)	<code>variables1.var = variables2.var</code>
Graph property(ies)	<code>NSCC RELOP LIMIT</code>
Example	$\text{nvalues} \left(\left(\begin{array}{c} \text{var} - 4, \\ \text{var} - 5, \\ \text{var} - 5, \\ \text{var} - 4, \\ \text{var} - 1, \\ \text{var} - 5 \end{array} \right), =, 3 \right)$ <p>Parts (A) and (B) of Figure 4.335 respectively show the initial and final graph. Since we use the <code>NSCC</code> graph property we show the different strongly connected components of the final graph. Each strongly connected component corresponds to one value which is assigned to some variables of the <code>VARIABLES</code> collection. The 3 following values 1, 4 and 5 are used by the variables of the <code>VARIABLES</code> collection.</p>
Usage	Used in the Constraint(s) on sets slot for defining some constraints like <code>assign_and_nvalues</code> , <code>circuit_cluster</code> or <code>coloured_cumulative</code> .
Used in	<code>assign_and_nvalues</code> , <code>circuit_cluster</code> , <code>coloured_cumulative</code> , <code>coloured_cumulatives</code> .
See also	<code>nvalues_except_0</code> , <code>nvalue</code> .
Key words	counting constraint, value partitioning constraint, number of distinct equivalence classes, number of distinct values, strongly connected component, equivalence.

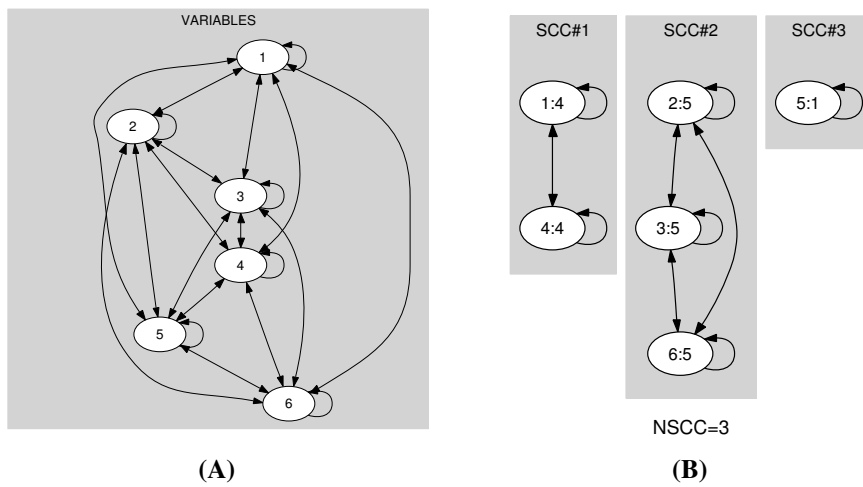


Figure 4.335: Initial and final graph of the nvalues constraint

4.157 nvalues_except_0

Origin	Derived from <code>nvalues</code> .
Constraint	<code>nvalues_except_0(VARIABLES, RELOP, LIMIT)</code>
Argument(s)	VARIABLES : collection(var – dvar) RELOP : atom LIMIT : dvar
Restriction(s)	required(VARIABLES, var) RELOP ∈ [=, ≠, <, ≥, >, ≤]
Purpose	Let N be the number of distinct values, different from 0, assigned to the variables of the <code>VARIABLES</code> collection. Enforce condition N RELOP LIMIT to hold.
Arc input(s)	VARIABLES
Arc generator	<i>CLIQUE</i> ↦ collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	<ul style="list-style-type: none"> • variables1.var ≠ 0 • variables1.var = variables2.var
Graph property(ies)	NSCC RELOP LIMIT
Example	$\text{nvalues_except_0} \left(\left(\begin{array}{c} \text{var} - 4, \\ \text{var} - 5, \\ \text{var} - 5, \\ \text{var} - 4, \\ \text{var} - 0, \\ \text{var} - 1 \end{array} \right), =, 3 \right)$ <p>Parts (A) and (B) of Figure 4.336 respectively show the initial and final graph. Since we use the NSCC graph property we show the different strongly connected components of the final graph. Each strongly connected component corresponds to one value distinct from 0 which is assigned to some variables of the <code>VARIABLES</code> collection. Beside value 0, the 3 following values 1, 4 and 5 are assigned to the variables of the <code>VARIABLES</code> collection.</p>
Used in	<code>cycle_or_accessibility</code> .
See also	<code>nvalues</code> , <code>nvalue</code> , <code>assign_and_nvalues</code> .
Key words	counting constraint, value partitioning constraint, number of distinct values, strongly connected component, joker value.

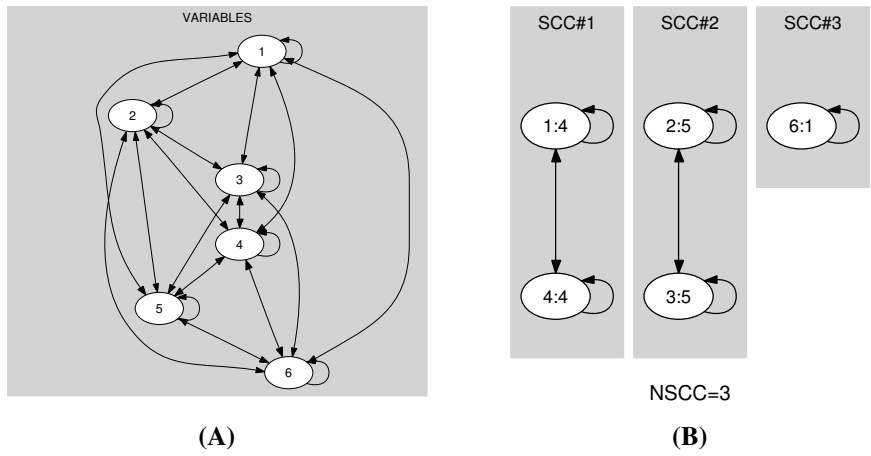


Figure 4.336: Initial and final graph of the `nvalues_except_0` constraint

4.158 one_tree

Origin	Inspired by [134]
Constraint	one_tree(NODES)
Argument(s)	$\text{NODES : collection} \left(\begin{array}{l} \text{id - atom,} \\ \text{index - int,} \\ \text{type - int,} \\ \text{father - dvar,} \\ \text{depth1 - dvar,} \\ \text{depth2 - dvar} \end{array} \right)$
Restriction(s)	<pre> required(NODES, [id, index, type, father, depth1, depth2]) NODES.index ≥ 1 NODES.index ≤ NODES distinct(NODES, index) in_list(NODES, type, [2, 3, 6]) NODES.father ≥ 1 NODES.father ≤ NODES NODES.depth1 ≥ 0 NODES.depth1 ≤ NODES NODES.depth2 ≥ 0 NODES.depth2 ≤ NODES </pre>
Purpose	Merge two trees that have some leaves in common so that all the precedence constraints induced by the father relation of both trees are preserved.
<hr/>	
Arc input(s)	NODES
Arc generator	<i>CLIQUE</i> ↦ collection(nodes1, nodes2)
Arc arity	2
Arc constraint(s)	$\bigvee \left(\bigwedge \left(\begin{array}{l} \text{nodes1.index = nodes2.index} \wedge \text{nodes1.father = nodes1.index,} \\ \text{nodes1.index} \neq \text{nodes2.index,} \\ \text{nodes1.father = nodes2.index,} \\ \bigvee \left(\begin{array}{l} \text{nodes1.type mod 2 = 0} \wedge \text{nodes1.depth1} > \text{nodes2.depth1,} \\ \text{nodes1.type mod 2} > 0 \wedge \text{nodes1.depth1} = \text{nodes2.depth1} \end{array} \right), \\ \bigvee \left(\begin{array}{l} \text{nodes1.type mod 3 = 0} \wedge \text{nodes1.depth2} > \text{nodes2.depth2,} \\ \text{nodes1.type mod 3} > 0 \wedge \text{nodes1.depth2} = \text{nodes2.depth2} \end{array} \right) \end{array} \right) \right)$
Graph property(ies)	<ul style="list-style-type: none"> • MAX_NSCC ≤ 1 • NCC = 1 • NVERTEX = NODES

Example

one_tree	{	id - x index - 1 type - 2 father - 6 depth1 - 2 depth2 - 2,
		id - x index - 2 type - 2 father - 2 depth1 - 1 depth2 - 0,
		id - x index - 3 type - 3 father - 6 depth1 - 1 depth2 - 3,
		id - x index - 4 type - 3 father - 5 depth1 - 2 depth2 - 4,
		id - x index - 5 type - 3 father - 1 depth1 - 2 depth2 - 3,
		id - x index - 6 type - 3 father - 7 depth1 - 1 depth2 - 2,
		id - x index - 7 type - 3 father - 2 depth1 - 1 depth2 - 1,
		id - g index - 8 type - 2 father - 1 depth1 - 3 depth2 - 2,
		id - a index - 9 type - 6 father - 4 depth1 - 3 depth2 - 5,
		id - f index - 10 type - 6 father - 7 depth1 - 2 depth2 - 2,
		id - b index - 11 type - 3 father - 4 depth1 - 2 depth2 - 5,
		id - c index - 12 type - 3 father - 5 depth1 - 2 depth2 - 4,
		id - e index - 13 type - 3 father - 3 depth1 - 1 depth2 - 4,
		id - d index - 14 type - 3 father - 3 depth1 - 1 depth2 - 4

Figure 4.337 shows the two trees we want to merge. Note that the leaves a and f occur in both trees. In order to ease the link with the merged tree given in part (B) of Figure 4.338, each vertex of the original trees contains the id, the index, the type, the father and the corresponding depth.

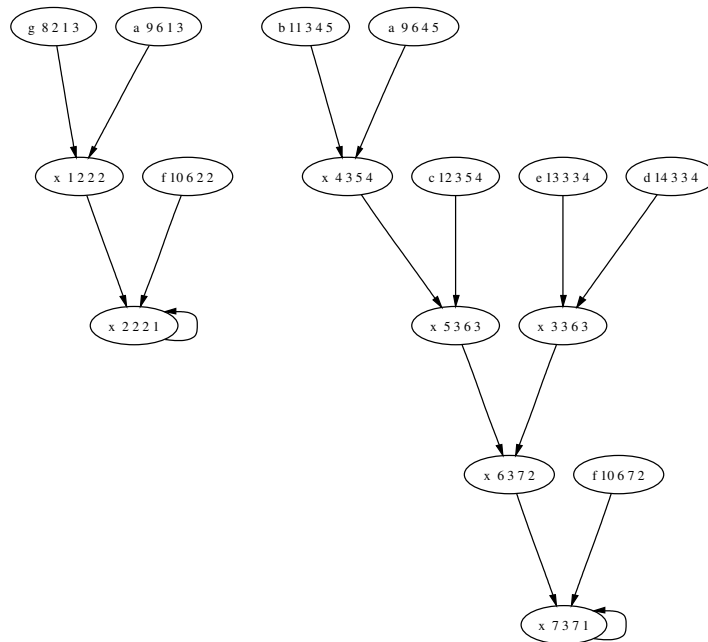
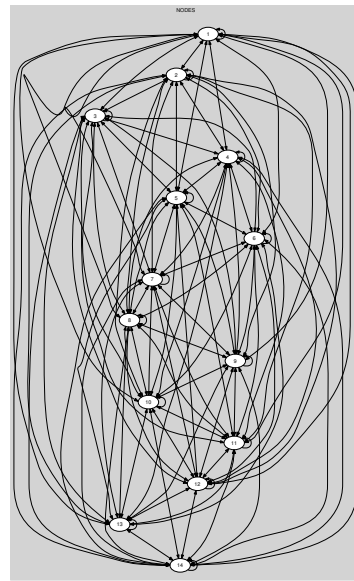


Figure 4.337: The two trees to merge

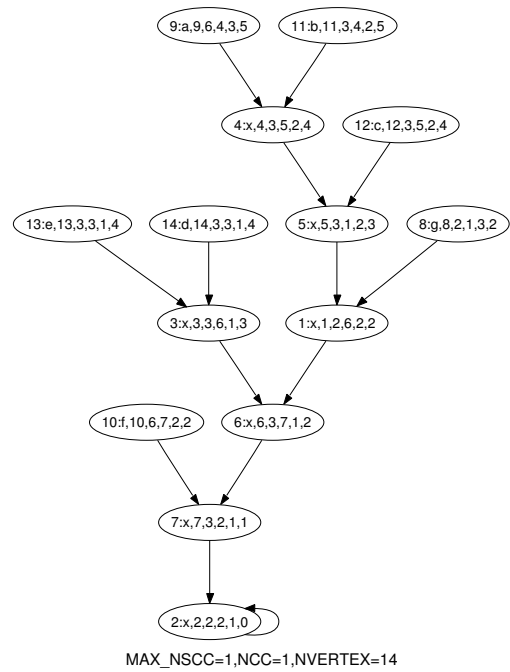
Parts (A) and (B) of Figure 4.338 respectively show the initial and final graph. Since we use the **NVERTEX** graph property, the vertices of the final graph are stressed in bold.

Graph model

The information about the two trees to merge is modelled in the following way:



(A)



(B)

Figure 4.338: Initial and final graph of the one_tree constraint

- A vertex which only belongs to the first (respectively second) tree has its `type` attribute set to 2 (respectively 3), while a vertex which belongs to both trees has its `type` attribute set to 6. This encoding was selected so that the statement `type mod 2 = 0` (respectively `type mod 3 = 0`) allows determining whether a vertex belongs or not to the first (respectively second) tree.
- For a vertex belonging to the first (respectively second) tree, the `depth1` (respectively `depth2`) attribute indicates the depth of that vertex in the corresponding tree.

The arc constraint is a disjunction of two conditions which respectively capture the following ideas:

- The first condition describes the fact that we link a vertex to itself. This vertex corresponds to the root of the merged tree we construct.
- The first part of the second condition describes the fact that we link a child vertex `nodes1` to its father `nodes2`. The last part of the second condition expresses the fact that we want to preserve the father relation imposed by the first and second trees. This is achieved by using the following idea: When the child vertex `nodes1` belongs to the first (respectively second) tree we enforce a strict inequality between the `depth1` (respectively `depth2`) attributes of `nodes1` and `nodes2`; Otherwise we enforce an equality constraint.

Finally we use the following three graph properties in order to enforce to get a merged tree:

- The first graph property $\text{MAX_NSCC} \leq 1$ enforces the fact that the size of the largest strongly connected component does not exceed one. This avoid having circuits containing more than one vertex. In fact the root of the merged tree is a strongly connected component with one single vertex.
- The second graph property $\text{NCC} = 1$ imposes having only one single tree.
- Finally the third graph property $\text{NVERTEX} = |\text{NODES}|$ imposes that the merged tree contains effectively all the vertices of the first and second tree.

Remark A compact way to model the construction of a *tree of life* [134].

See also `tree`.

Key words graph constraint, tree, bioinformatics, phylogeny, obscure.

4.159 orchard

Origin	[135]
Constraint	orchard(NROW, TREES)
Argument(s)	NROW : dvar TREES : collection(index - int, x - dvar, y - dvar)
Restriction(s)	NROW ≥ 0 TREES.index ≥ 1 TREES.index ≤ TREES required(TREES, [index, x, y]) distinct(TREES, index) TREES.x ≥ 0 TREES.y ≥ 0
Purpose	Orchard problem [135]: Your aid I want, Nine trees to plant, In rows just half a score, And let there be, In each row, three—Solve this: I ask no more!
Arc input(s)	TREES
Arc generator	<i>CLIQUE</i> (<) ↦ collection(trees1, trees2, trees3)
Arc arity	3
Arc constraint(s)	$\sum \begin{pmatrix} \text{trees1.x} * \text{trees2.y} - \text{trees1.x} * \text{trees3.y}, \\ \text{trees1.y} * \text{trees3.x} - \text{trees1.y} * \text{trees2.x}, \\ \text{trees2.x} * \text{trees3.y} - \text{trees2.y} * \text{trees3.x} \end{pmatrix} = 0$
Graph property(ies)	NARC = NROW

Example	orchard $\left(10, \left\{ \begin{array}{l} \text{index} - 1 \quad x - 0 \quad y - 0, \\ \text{index} - 2 \quad x - 4 \quad y - 0, \\ \text{index} - 3 \quad x - 8 \quad y - 0, \\ \text{index} - 4 \quad x - 2 \quad y - 4, \\ \text{index} - 5 \quad x - 4 \quad y - 4, \\ \text{index} - 6 \quad x - 6 \quad y - 4, \\ \text{index} - 7 \quad x - 0 \quad y - 8, \\ \text{index} - 8 \quad x - 4 \quad y - 8, \\ \text{index} - 9 \quad x - 8 \quad y - 8 \end{array} \right\} \right)$
----------------	---

The 10 alignments of 3 trees correspond to the following triples of trees: (1, 2, 3), (1, 4, 8), (1, 5, 9), (2, 4, 7), (2, 5, 8), (2, 6, 9), (3, 5, 7), (3, 6, 8), (4, 5, 6), (7, 8, 9). Figure 4.339 shows the 9 trees and the 10 alignments corresponding to the example.

Graph model

The arc generator $CLIQUE(<)$ with an arity of three is used in order to generate all the arcs of the directed hypergraph. Each arc is an ordered triple of trees. We use the restriction $<$ in order to generate one single arc for each set of three trees. This is required, since otherwise we would count more than once a given alignment of three trees. The formula used within the arc constraint expresses the fact that the three points of respective coordinates $(trees_1.x, trees_1.y)$, $(trees_2.x, trees_2.y)$ and $(trees_3.x, trees_3.y)$ are aligned. It corresponds to the development of the expression:

$$\begin{vmatrix} trees_1.x & trees_2.y & 1 \\ trees_2.x & trees_2.y & 1 \\ trees_3.x & trees_3.y & 1 \end{vmatrix} = 0$$

Key words

geometrical constraint, alignment, hypergraph.

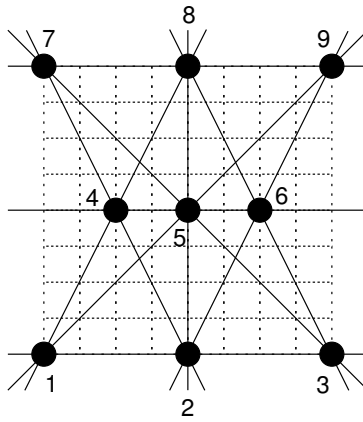


Figure 4.339: Nine trees with 10 alignments of 3 trees

20000128

715

4.160 `orth_link_ori_siz_end`

Origin	Used by several constraints between orthotopes
Constraint	<code>orth_link_ori_siz_end(ORTHOTOPE)</code>
Argument(s)	<code>ORTHOTOPE : collection(ori - dvar, siz - dvar, end - dvar)</code>
Restriction(s)	<code> ORTHOTOPE > 0</code> <code>require_at_least(2, ORTHOTOPE, [ori, siz, end])</code> <code>ORTHOTOPE.siz ≥ 0</code>
Purpose	Enforce for each item of the ORTHOTOPE collection the constraint <code>ori + siz = end</code> .
Arc input(s)	ORTHOTOPE
Arc generator	<code>SELF ↦ collection(orthotope)</code>
Arc arity	1
Arc constraint(s)	<code>orthotope.ori + orthotope.siz = orthotope.end</code>
Graph property(ies)	<code>NARC = ORTHOTOPE </code>

Example `orth_link_ori_siz_end` $\left(\left\{ \begin{array}{ccc} \text{ori} - 2 & \text{siz} - 2 & \text{end} - 4, \\ \text{ori} - 1 & \text{siz} - 3 & \text{end} - 4 \end{array} \right\} \right)$

Parts (A) and (B) of Figure 4.340 respectively show the initial and final graph. Since we use the **NARC** graph property, the unary arcs of the final graph are stressed in bold.



Figure 4.340: Initial and final graph of the `orth_link_ori_siz_end` constraint

Signature	Since we use the <i>SELF</i> arc generator on the ORTHOTOPE collection the number of arcs of the initial graph is equal to $ \text{ORTHOTOPE} $. Therefore the maximum number of arcs of the final graph is also equal to $ \text{ORTHOTOPE} $. For this reason we can rewrite the graph property $\text{NARC} = \text{ORTHOTOPE} $ to $\text{NARC} \geq \text{ORTHOTOPE} $ and simplify $\overline{\text{NARC}}$ to $\overline{\text{NARC}}$.
Usage	Used in the Arc constraint(s) slot for defining some constraints like <code>diffn</code> , <code>place_in_pyramid</code> or <code>orths_are_connected</code> .

Used in diffn, orth_on_the_ground, orth_on_top_of_orth, orths_are_connected,
two_orth_are_in_contact, two_orth_column, two_orth_do_not_overlap,
two_orth_include.

Key words decomposition, orthotope.

4.161 orth_on_the_ground

Origin	Used for defining place_in_pyramid.
Constraint	orth_on_the_ground(ORTHOTOPE, VERTICAL_DIM)
Argument(s)	ORTHOTOPE : collection(ori - dvar, siz - dvar, end - dvar) VERTICAL_DIM : int
Restriction(s)	ORTHOTOPE > 0 require_at_least(2, ORTHOTOPE, [ori, siz, end]) ORTHOTOPE.siz ≥ 0 VERTICAL_DIM ≥ 1 VERTICAL_DIM ≤ ORTHOTOPE orth_link_ori_siz_end(ORTHOTOPE)
Purpose	The ori attribute of the VERTICAL_DIM th item of the ORTHOTOPES collection should be fixed to one.
Arc input(s)	ORTHOTOPE
Arc generator	SELF ↦ collection(orthotope)
Arc arity	1
Arc constraint(s)	<ul style="list-style-type: none"> • orthotope.key = VERTICAL_DIM • orthotope.ori = 1
Graph property(ies)	NARC = 1

Example $\text{orth_on_the_ground} \left(\left\{ \begin{array}{ccc} \text{ori} - 1 & \text{siz} - 2 & \text{end} - 3, \\ \text{ori} - 2 & \text{siz} - 3 & \text{end} - 5 \end{array} \right\}, 1 \right)$

Parts (A) and (B) of Figure 4.341 respectively show the initial and final graph. Since we use the **NARC** graph property, the unary arc of the final graph is stressed in bold.

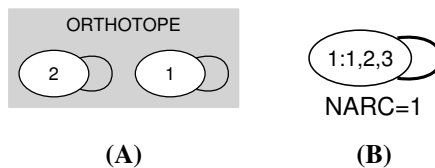


Figure 4.341: Initial and final graph of the orth_on_the_ground constraint

Signature	Since all the key attributes of the ORTHOTOPES collection are distinct, because of the first condition of the arc constraint, and since we use the <i>SELF</i> arc generator the final graph contains at most one arc. Therefore we can rewrite the graph property $\text{NARC} = 1$ to $\text{NARC} \geq 1$ and simplify <u>NARC</u> to $\overline{\text{NARC}}$.
Used in	place_in_pyramid.
See also	place_in_pyramid.
Key words	geometrical constraint, orthotope.

4.162 orth_on_top_of_orth

Origin	Used for defining <code>place_in_pyramid</code> .
Constraint	<code>orth_on_top_of_orth(ORTHOTOPE1, ORTHOTOPE2, VERTICAL_DIM)</code>
Type(s)	<code>ORTHOTOPE : collection(ori - dvar, siz - dvar, end - dvar)</code>
Argument(s)	<code>ORTHOTOPE1 : ORTHOTOPE</code> <code>ORTHOTOPE2 : ORTHOTOPE</code> <code>VERTICAL_DIM : int</code>
Restriction(s)	<code> ORTHOTOPE > 0</code> <code>require_at_least(2, ORTHOTOPE, [ori, siz, end])</code> <code>ORTHOTOPE.siz ≥ 0</code> <code> ORTHOTOPE1 = ORTHOTOPE2 </code> <code>VERTICAL_DIM ≥ 1</code> <code>VERTICAL_DIM ≤ ORTHOTOPE1 </code> <code>orth_link_ori_siz_end(ORTHOTOPE1)</code> <code>orth_link_ori_siz_end(ORTHOTOPE2)</code>
Purpose	<p>ORTHOTOPE1 is located on top of ORTHOTOPE2 which concretely means:</p> <ul style="list-style-type: none"> • In each dimension different from <code>VERTICAL_DIM</code> the projection of ORTHOTOPE1 is included in the projection of ORTHOTOPE2. • In the dimension <code>VERTICAL_DIM</code> the origin of ORTHOTOPE1 coincide with the end of ORTHOTOPE2.
Arc input(s)	<code>ORTHOTOPE1 ORTHOTOPE2</code>
Arc generator	<code>PRODUCT(=) ↦ collection(orthotope1, orthotope2)</code>
Arc arity	2
Arc constraint(s)	<ul style="list-style-type: none"> • <code>orthotope1.key ≠ VERTICAL_DIM</code> • <code>orthotope2.ori ≤ orthotope1.ori</code> • <code>orthotope1.end ≤ orthotope2.end</code>
Graph property(ies)	<code>NARC = ORTHOTOPE1 - 1</code>
Arc input(s)	<code>ORTHOTOPE1 ORTHOTOPE2</code>
Arc generator	<code>PRODUCT(=) ↦ collection(orthotope1, orthotope2)</code>
Arc arity	2
Arc constraint(s)	<ul style="list-style-type: none"> • <code>orthotope1.key = VERTICAL_DIM</code> • <code>orthotope1.ori = orthotope2.end</code>

Graph property(ies) **NARC = 1**

Example $\text{orth_on_top_of_orth} \left(\left(\left\{ \begin{array}{l} \text{ori} - 5 \quad \text{siz} - 2 \quad \text{end} - 7, \\ \text{ori} - 3 \quad \text{siz} - 3 \quad \text{end} - 6, \\ \text{ori} - 3 \quad \text{siz} - 5 \quad \text{end} - 8, \\ \text{ori} - 1 \quad \text{siz} - 2 \quad \text{end} - 3 \end{array} \right\}, 2 \right) \right)$

Parts (A) and (B) of Figure 4.342 respectively show the initial and final graph associated to the second graph constraint. Since we use the **NARC** graph property, the unique arc of the final graph is stressed in bold.

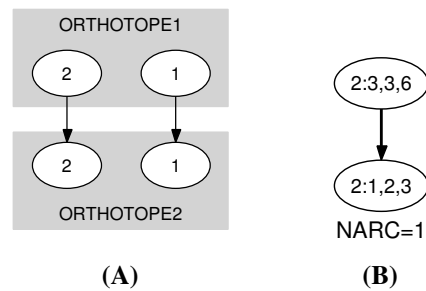


Figure 4.342: Initial and final graph of the `orth_on_top_of_orth` constraint

Graph model The first and second graph constraints respectively express the first and second conditions stated in the **Purpose** slot defining the `orth_on_top_of_orth` constraint.

Signature Consider the second graph constraint. Since all the key attributes of the `ORTHOTOPE1` collection are distinct, because of the arc constraint `orthotope1.key = VERTICAL_DIM`, and since we use the *PRODUCT(=)* arc generator the final graph contains at most one arc. Therefore we can rewrite the graph property `NARC = 1` to `NARC ≥ 1` and simplify NARC to NARC.

Used in `place_in_pyramid`.

See also `place_in_pyramid`.

Key words geometrical constraint, non-overlapping, orthotope.

4.163 `orths_are_connected`

Origin	N. Beldiceanu
Constraint	<code>orths_are_connected(ORTHOTOPE)</code>
Type(s)	<code>ORTHOTOPE</code> : <code>collection(ori - dvar, siz - dvar, end - dvar)</code>
Argument(s)	<code>ORTHOTOPE</code> : <code>collection(orth - ORTHOTOPE)</code>
Restriction(s)	<code> ORTHOTOPE > 0</code> <code>require_at_least(2, ORTHOTOPE, [ori, siz, end])</code> <code>ORTHOTOPE.siz > 0</code> <code>required(ORTHOTOPE, orth)</code> <code>same_size(ORTHOTOPE, orth)</code>

Purpose

There should be one single group of connected orthotopes. Two orthotopes touch each other (i.e. are connected) if they overlap in all dimensions except one, and if, for the dimension where they do not overlap, the distance between the two orthotopes is equal to 0.

Arc input(s)	<code>ORTHOTOPE</code>
Arc generator	$\text{SELF} \mapsto \text{collection}(\text{orthotopes})$
Arc arity	1
Arc constraint(s)	<code>orth_link_ori_siz_end(orthotopes.orth)</code>
Graph property(ies)	$\text{NARC} = \text{ORTHOTOPE} $

Arc input(s)	<code>ORTHOTOPE</code>
Arc generator	$\text{CLIQUE}(\neq) \mapsto \text{collection}(\text{orthotopes1}, \text{orthotopes2})$
Arc arity	2
Arc constraint(s)	<code>two_orth_are_in_contact(orthotopes1.orth, orthotopes2.orth)</code>
Graph property(ies)	<ul style="list-style-type: none"> • $\text{NVERTEX} = \text{ORTHOTOPE}$ • $\text{NCC} = 1$

Example

`orths_are_connected` $\left(\left(\left(\begin{array}{l} \text{orth} - \left\{ \begin{array}{l} \text{ori} - 2 \quad \text{siz} - 4 \quad \text{end} - 6, \\ \text{ori} - 2 \quad \text{siz} - 2 \quad \text{end} - 4 \end{array} \right\}, \\ \text{orth} - \left\{ \begin{array}{l} \text{ori} - 1 \quad \text{siz} - 2 \quad \text{end} - 3, \\ \text{ori} - 4 \quad \text{siz} - 3 \quad \text{end} - 7 \end{array} \right\}, \\ \text{orth} - \left\{ \begin{array}{l} \text{ori} - 7 \quad \text{siz} - 4 \quad \text{end} - 11, \\ \text{ori} - 1 \quad \text{siz} - 2 \quad \text{end} - 3 \end{array} \right\}, \\ \text{orth} - \left\{ \begin{array}{l} \text{ori} - 6 \quad \text{siz} - 2 \quad \text{end} - 8, \\ \text{ori} - 3 \quad \text{siz} - 2 \quad \text{end} - 5 \end{array} \right\} \end{array} \right) \right) \right)$

Parts (A) and (B) of Figure 4.343 respectively show the initial and final graph. Since we use the **NVERTEX** graph property the vertices of the final graph are stressed in bold. Since we also use the **NCC** graph property we show the unique connected component of the final graph. An arc between two vertices indicates that two rectangles are in contact. Figure 4.344 shows the rectangles associated to the example. One can observe that:

- Rectangle 2 touch rectangle 1,
- Rectangle 1 touch rectangle 2 and rectangle 4,
- Rectangle 4 touch rectangle 1 and rectangle 3,
- Rectangle 3 touch rectangle 4.

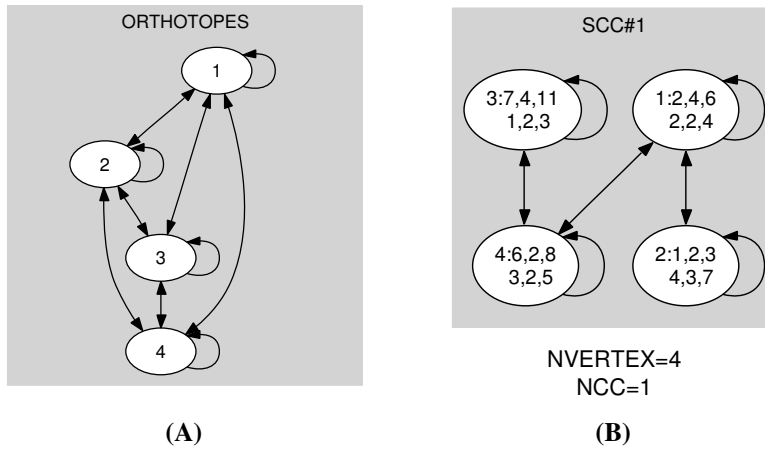


Figure 4.343: Initial and final graph of the orths_are_connected constraint

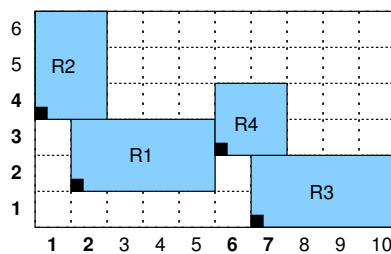


Figure 4.344: Four connected rectangles

Signature

Since the first graph constraint uses the *SELF* arc generator on the ORTHOTOPES collection the corresponding initial graph contains $|\text{ORTHOTOPES}|$ arcs. Therefore the final graph of the first graph constraint contains at most $|\text{ORTHOTOPES}|$ arcs and we can rewrite $\text{NARC} = |\text{ORTHOTOPES}|$ to $\text{NARC} \geq |\text{ORTHOTOPES}|$. So we can simplify NARC to NARC.

Consider now the second graph constraint. Since its corresponding initial graph contains $|\text{ORTHOTOPES}|$ vertices, its final graph has a maximum number of vertices also equal to $|\text{ORTHOTOPES}|$. Therefore we can rewrite $\text{NVERTEX} = |\text{ORTHOTOPES}|$ to $\text{NVERTEX} \geq |\text{ORTHOTOPES}|$ and simplify $\overline{\text{NVERTEX}}$ to $\overline{\text{NVERTEX}}$. From the graph property $\text{NVERTEX} = |\text{ORTHOTOPES}|$ and from the restriction $|\text{ORTHOTOPES}| > 0$ the final graph is not empty. Therefore it contains at least one connected component. So we can rewrite $\text{NCC} = 1$ to $\text{NCC} \leq 1$ and simplify $\overline{\text{NCC}}$ to $\overline{\text{NCC}}$.

Usage	In floor planning problem there is a typical constraint, which states that one should be able to access every room from any room.
See also	<code>two_orth_are_in_contact</code> .
Key words	geometrical constraint, touch, contact, non-overlapping, orthotope.

20000128

725

4.164 path_from_to

Origin	[74]
Constraint	path_from_to(FROM, TO, NODES)
Usual name	path
Argument(s)	FROM : int TO : int NODES : collection(index – int, succ – svar)
Restriction(s)	FROM ≥ 1 FROM ≤ NODES TO ≥ 1 TO ≤ NODES required(NODES, [index, succ]) NODES.index ≥ 1 NODES.index ≤ NODES distinct(NODES, index)
Purpose	Select some arcs of a digraph G so that there is still a path between two given vertices of G .
Arc input(s)	NODES
Arc generator	CLIQUE \mapsto collection(nodes1, nodes2)
Arc arity	2
Arc constraint(s)	in_set(nodes2.index, nodes1.succ)
Graph property(ies)	<u>PATH_FROM_TO(index, FROM, TO) = 1</u>

Example $\text{path_from_to} \left(4, 3, \left\{ \begin{array}{l} \text{index} - 1 \quad \text{succ} - \emptyset, \\ \text{index} - 2 \quad \text{succ} - \emptyset, \\ \text{index} - 3 \quad \text{succ} - \{5\}, \\ \text{index} - 4 \quad \text{succ} - \{5\}, \\ \text{index} - 5 \quad \text{succ} - \{2, 3\} \end{array} \right\} \right)$

Part (A) of Figure 4.345 shows the initial graph from which we choose to start. It is derived from the set associated to each vertex. Each set describes the potential values of the succ attribute of a given vertex. Part (B) of Figure 4.345 gives the final graph associated to the example. Since we use the **PATH_FROM_TO** graph property we show on the final graph the following information:

- The vertices which respectively correspond to the start and the end of the required path are stressed in bold.
- The arcs on the required path are also stressed in bold.

The `path_from_to` constraint holds since there is a path from vertex 4 to vertex 3 (4 and 3 refer to the `index` attribute of a vertex).

Signature

Since the maximum value returned by the graph property `PATH_FROM_TO` is equal to 1 we can rewrite `PATH_FROM_TO(index, FROM, TO) = 1` to `PATH_FROM_TO(index, FROM, TO) ≥ 1`. Therefore we simplify `PATH_FROM_TO` to `PATH_FROM_TO`.

See also

`temporal_path`, `link_set_to_booleans`.

Key words

graph constraint, path, linear programming, constraint involving set variables.

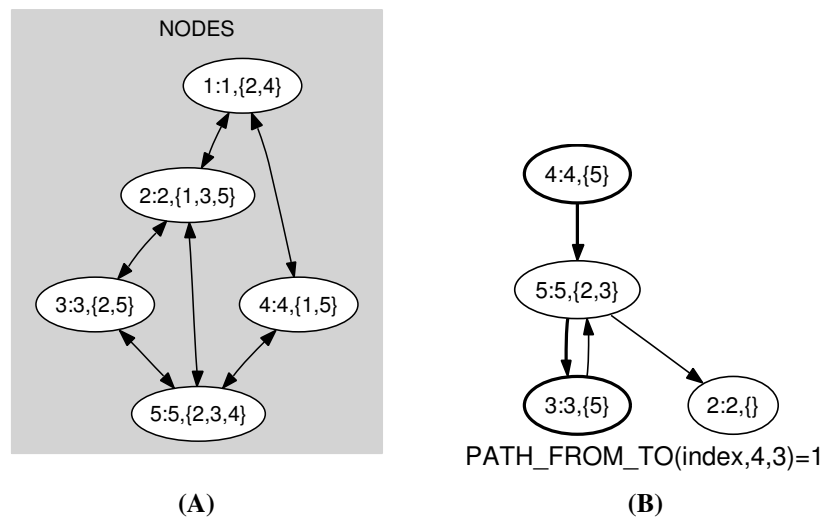


Figure 4.345: Initial and final graph of the path_from_to set constraint

20030820

729

4.165 pattern

Origin	[34]
Constraint	pattern(VARIABLES, PATTERNS)
Type(s)	PATTERN : collection(var – int)
Argument(s)	VARIABLES : collection(var – dvar) PATTERNS : collection(pat – PATTERN)
Restriction(s)	required(PATTERN, var) change(0, PATTERN, =) required(VARIABLES, var) required(PATTERNS, pat) same_size(PATTERNS, pat)

Purpose

We quote the definition from the original paper [34, page 157] introducing the pattern constraint.

We call a k -pattern any sequence of k elements such that no two successive elements have the same value. Consider a set $V = \{v_1, v_2, \dots, v_m\}$ and a sequence $s = \langle s_1, s_2, \dots, s_n \rangle$ of elements of V . Consider now the sequence $\langle v_{i_1}, v_{i_2}, \dots, v_{i_l} \rangle$ of the types of the successive stretches that appear in s . Let \mathcal{P} be a set of k -pattern. Vector s satisfies \mathcal{P} if and only if every subsequence of k elements in $\langle v_{i_1}, v_{i_2}, \dots, v_{i_l} \rangle$ belongs to \mathcal{P} .

Example

$$\text{pattern} \left(\left(\begin{array}{l} \text{var} - 1, \\ \text{var} - 1, \\ \text{var} - 2, \\ \text{var} - 2, \\ \text{var} - 2, \\ \text{var} - 1, \\ \text{var} - 3, \\ \text{var} - 3 \end{array} \right), \left(\begin{array}{l} \text{pat} - \{\text{var} - 1, \text{var} - 2, \text{var} - 1\}, \\ \text{pat} - \{\text{var} - 1, \text{var} - 2, \text{var} - 3\}, \\ \text{pat} - \{\text{var} - 2, \text{var} - 1, \text{var} - 3\} \end{array} \right) \right)$$

Usage

The pattern constraint was originally introduced within the context of staff scheduling. In this context, the value of the i^{th} variable of the VARIABLES collection corresponds to the type of shift performed by a person on the i^{th} day. A *stretch* is a maximum sequence of consecutive variables which are all assigned to the same value. The pattern constraint imposes that each sequence of k consecutive stretches belongs to a given list of patterns.

Remark

A generalization of the pattern constraint to the regular constraint enforcing the fact that a sequence of variables corresponds to a regular expression is presented in [5].

See also

stretch_path, sliding_distribution, group.

Key words

predefined constraint, timetabling constraint, sliding sequence constraint.

4.166 peak

Origin	Derived from <code>inflexion</code> .
Constraint	<code>peak(N, VARIABLES)</code>
Argument(s)	<code>N</code> : <code>dvar</code> <code>VARIABLES</code> : <code>collection(var - dvar)</code>
Restriction(s)	$N \geq 0$ $2 * N \leq \max(VARIABLES - 1, 0)$ <code>required(VARIABLES, var)</code>

Purpose

A variable V_k ($1 < k < m$) of the sequence of variables $VARIABLES = V_1, \dots, V_m$ is a *peak* if and only if there exist an i ($1 < i \leq k$) such that $V_{i-1} < V_i$ and $V_i = V_{i+1} = \dots = V_k$ and $V_k > V_{k+1}$. N is the total number of peaks of the sequence of variables $VARIABLES$.

Example

$$\text{peak} \left(2, \left\{ \begin{array}{l} \text{var} - 1, \\ \text{var} - 1, \\ \text{var} - 4, \\ \text{var} - 8, \\ \text{var} - 6, \\ \text{var} - 2, \\ \text{var} - 7, \\ \text{var} - 1 \end{array} \right\} \right)$$

The previous constraint holds since the sequence 1 1 4 8 6 2 7 1 contains two peaks which correspond to the variables which are assigned to values 8 and 7.

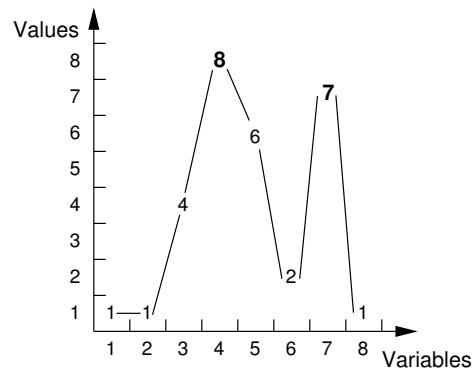


Figure 4.346: The sequence and its two peaks

Automaton

Figure 4.347 depicts the automaton associated to the `peak` constraint. To each pair of consecutive variables (VAR_i, VAR_{i+1}) of the collection $VARIABLES$ corresponds a signature variable S_i . The following signature constraint links VAR_i, VAR_{i+1} and S_i : $(VAR_i > VAR_{i+1} \Leftrightarrow S_i = 0) \wedge (VAR_i = VAR_{i+1} \Leftrightarrow S_i = 1) \wedge (VAR_i < VAR_{i+1} \Leftrightarrow S_i = 2)$.

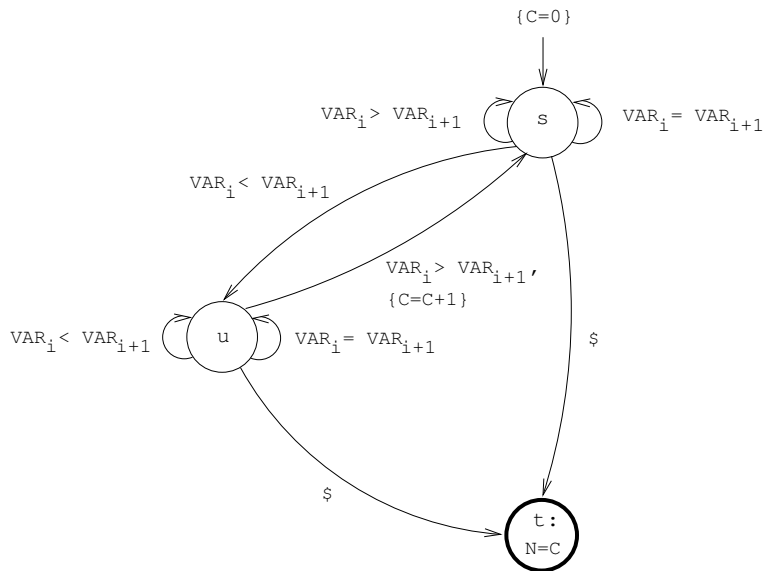


Figure 4.347: Automaton of the peak constraint

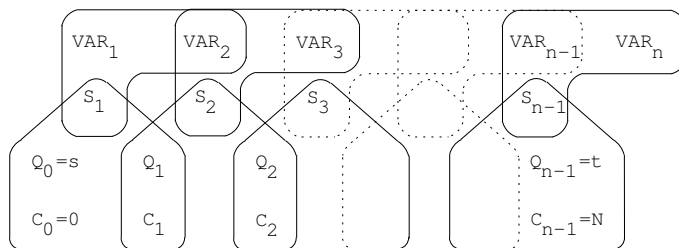


Figure 4.348: Hypergraph of the reformulation corresponding to the automaton of the peak constraint

Usage	Useful for constraining the number of <i>peaks</i> of a sequence of domain variables.
Remark	Since the arity of the arc constraint is not fixed, the <code>peak</code> constraint cannot be currently described. However, this would not hold anymore if we were introducing a slot that specifies how to merge adjacent vertices of the final graph.
See also	<code>no_peak</code> , <code>inflexion</code> , <code>valley</code> .
Key words	sequence, automaton, automaton with counters, sliding cyclic(1) constraint network(2).

20040530

735

4.167 period

Origin	N. Beldiceanu
Constraint	<code>period(PERIOD, VARIABLES, CTR)</code>
Argument(s)	<p><code>PERIOD</code> : <code>dvar</code> <code>VARIABLES</code> : <code>collection(var - dvar)</code> <code>CTR</code> : <code>atom</code></p>
Restriction(s)	<p>$PERIOD \geq 1$ $PERIOD \leq VARIABLES$ <code>required(VARIABLES, var)</code> $CTR \in [=, \neq, <, \geq, >, \leq]$</p>

Purpose

Let us note V_0, V_1, \dots, V_{m-1} the variables of the `VARIABLES` collection. `PERIOD` is the *period* of the sequence $V_0 V_1 \dots V_{m-1}$ according to constraint `CTR`. This means that `PERIOD` is the smallest natural number such that $V_i \text{ CTR } V_{i+\text{PERIOD}}$ holds for all $i \in \{0, 1, \dots, m - \text{PERIOD} - 1\}$.

Example

$$\text{period} \left(3, \left\{ \begin{array}{l} \text{var} - 1, \\ \text{var} - 1, \\ \text{var} - 4, \\ \text{var} - 1, \\ \text{var} - 1, \\ \text{var} - 4, \\ \text{var} - 1, \\ \text{var} - 1 \end{array} \right\}, = \right)$$

The smallest period of the previous sequence is equal to 3.

Algorithm When `CTR` corresponds to the equality constraint, a potentially incomplete filtering algorithm based on 13 deductions rules is described in [136]. The generalization of these rules to the case where `CTR` is not the equality constraint is discussed.

See also `period_except_0`.

Key words predefined constraint, periodic, timetabling constraint, scheduling constraint, sequence, border.

20000128

737

4.168 period_except_0

Origin Derived from `period`.

Constraint `period_except_0(PERIOD, VARIABLES, CTR)`

Argument(s)

PERIOD : dvar
 VARIABLES : collection(var – dvar)
 CTR : atom

Restriction(s)

PERIOD ≥ 1
 PERIOD $\leq |\text{VARIABLES}|$
 required(VARIABLES, var)
 CTR $\in [=, \neq, <, \geq, >, \leq]$

Purpose

Let us note V_0, V_1, \dots, V_{m-1} the variables of the VARIABLES collection. PERIOD is the *period* of the sequence $V_0 V_1 \dots V_{m-1}$ according to constraint CTR. This means that PERIOD is the smallest natural number such that $V_i \text{ CTR } V_{i+\text{PERIOD}} \vee V_i = 0 \vee V_{i+\text{PERIOD}} = 0$ holds for all $i \in 0, 1, \dots, m - \text{PERIOD} - 1$.

Example

$$\text{period_except_0} \left(3, \left\{ \begin{array}{l} \text{var} - 1, \\ \text{var} - 1, \\ \text{var} - 4, \\ \text{var} - 1, \\ \text{var} - 1, \\ \text{var} - 0, \\ \text{var} - 1, \\ \text{var} - 1 \end{array} \right\}, = \right)$$

Since value 0 is considered as a joker the fact that 4 is different from 0 does not matter. Therefore, the smallest period of the previous sequence is equal to 3.

Usage Useful for timetabling problems where a person should repeat some work pattern over an over except when he is unavailable for some reason. The value 0 represents the fact that he is unavailable, while the other values are used in the work pattern.

Algorithm See [136].

See also `period`.

Key words predefined constraint, periodic, timetabling constraint, scheduling constraint, sequence, joker value.

20030820

739

4.169 place_in_pyramid

Origin	N. Beldiceanu
Constraint	place_in_pyramid(ORTHOTOPES, VERTICAL_DIM)
Type(s)	ORTHOTOPE : collection(ori - dvar, siz - dvar, end - dvar)
Argument(s)	ORTHOTOPES : collection(orth - ORTHOTOPE) VERTICAL_DIM : int
Restriction(s)	ORTHOTOPE > 0 require_at_least(2, ORTHOTOPE, [ori, siz, end]) ORTHOTOPE.siz ≥ 0 same_size(ORTHOTOPES, orth) VERTICAL_DIM ≥ 1 diffn(ORTHOTOPES)
Purpose	For each pair of orthotopes (O_1, O_2) of the collection ORTHOTOPES, O_1 and O_2 do not overlap (two orthotopes do not overlap if there exists at least one dimension where their projections do not overlap). In addition, each orthotope of the collection ORTHOTOPES should be supported by one other orthotope or by the ground. The vertical dimension is given by the parameter VERTICAL_DIM.
Arc input(s)	ORTHOTOPES
Arc generator	CLIQUE \mapsto collection(orthotopes1, orthotopes2)
Arc arity	2
Arc constraint(s)	$\bigvee \left(\begin{array}{l} \bigwedge \left(\begin{array}{l} \text{orthotopes1.key} = \text{orthotopes2.key}, \\ \text{orth_on_the_ground}(\text{orthotopes1.orth}, \text{VERTICAL_DIM}) \end{array} \right), \\ \bigwedge \left(\begin{array}{l} \text{orthotopes1.key} \neq \text{orthotopes2.key}, \\ \text{orth_on_top_of_orth}(\text{orthotopes1.orth}, \text{orthotopes2.orth}, \text{VERTICAL_DIM}) \end{array} \right) \end{array} \right)$
Graph property(ies)	NARC = ORTHOTOPES

Example	$\text{place_in_pyramid} \left(\left(\left(\begin{array}{l} \text{orth} - \left\{ \begin{array}{l} \text{ori} - 1 \quad \text{siz} - 3 \quad \text{end} - 4, \\ \text{ori} - 1 \quad \text{siz} - 2 \quad \text{end} - 3 \end{array} \right\}, \\ \text{orth} - \left\{ \begin{array}{l} \text{ori} - 1 \quad \text{siz} - 2 \quad \text{end} - 3, \\ \text{ori} - 3 \quad \text{siz} - 3 \quad \text{end} - 6 \end{array} \right\}, \\ \text{orth} - \left\{ \begin{array}{l} \text{ori} - 5 \quad \text{siz} - 6 \quad \text{end} - 11, \\ \text{ori} - 1 \quad \text{siz} - 2 \quad \text{end} - 3 \end{array} \right\}, \\ \text{orth} - \left\{ \begin{array}{l} \text{ori} - 5 \quad \text{siz} - 2 \quad \text{end} - 7, \\ \text{ori} - 3 \quad \text{siz} - 2 \quad \text{end} - 5 \end{array} \right\}, \\ \text{orth} - \left\{ \begin{array}{l} \text{ori} - 8 \quad \text{siz} - 3 \quad \text{end} - 11, \\ \text{ori} - 3 \quad \text{siz} - 2 \quad \text{end} - 5 \end{array} \right\}, \\ \text{orth} - \left\{ \begin{array}{l} \text{ori} - 8 \quad \text{siz} - 2 \quad \text{end} - 10, \\ \text{ori} - 5 \quad \text{siz} - 2 \quad \text{end} - 7 \end{array} \right\} \end{array} \right), 2 \right)$
----------------	---

Parts (A) and (B) of Figure 4.349 respectively show the initial and final graph. Since we use the **NARC** graph property, the arcs of the final graph are stressed in bold. Figure 4.350 depicts the placement associated to the example.

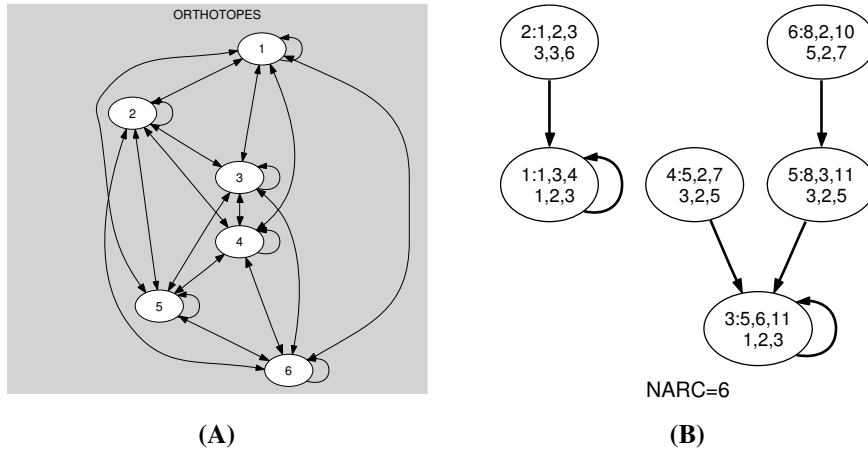


Figure 4.349: Initial and final graph of the `place_in_pyramid` constraint

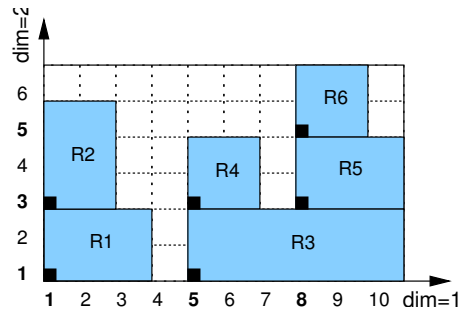


Figure 4.350: Solution corresponding to the final graph

Graph model

The arc constraint of the graph constraint enforces one of the following conditions:

- If the arc connects the same orthotope O then the ground directly supports O ,
- Otherwise, if we have an arc from a orthotope O_1 to a distinct orthotope O_2 , the condition is: O_1 is on top of O_2 (i.e. in all dimensions, except dimension `VERTICAL_DIM`, the projection of O_1 is included in the projection of O_2 , while in dimension `VERTICAL_DIM` the projection of O_1 is located after the projection of O_2).

Usage

The `diffn` constraint is not enough if one wants to produce a placement where no orthotope floats in the air. This constraint is usually handled with a heuristic during the enumeration phase.

See also

`orth_on_top_of_orth`, `orth_on_the_ground`.

Key words

geometrical constraint, non-overlapping, orthotope.

20000128

743

4.170 polyomino

Origin	Inspired by [137].
Constraint	polyomino(CELLS)
Argument(s)	CELLS : collection(index – int, right – dvar, left – dvar, up – dvar, down – dvar)
Restriction(s)	CELLS.index ≥ 1 CELLS.index $\leq \text{CELLS} $ $ \text{CELLS} \geq 1$ required(CELLS, [index, right, left, up, down]) distinct(CELLS, index) CELLS.right ≥ 0 CELLS.right $\leq \text{CELLS} $ CELLS.left ≥ 0 CELLS.left $\leq \text{CELLS} $ CELLS.up ≥ 0 CELLS.up $\leq \text{CELLS} $ CELLS.down ≥ 0 CELLS.down $\leq \text{CELLS} $
Purpose	<p>Enforce all cells of the collection CELLS to be connected. Each cell is defined by the following attributes:</p> <ol style="list-style-type: none"> 1. The <code>index</code> attribute of the cell, which is an integer between 1 and the total number of cells, is unique for each cell. 2. The <code>right</code> attribute, which is the index of the cell located immediately to the right of that cell (or 0 if no such cell exists). 3. The <code>left</code> attribute, which is the index of the cell located immediately to the left of that cell (or 0 if no such cell exists). 4. The <code>up</code> attribute, which is the index of the cell located immediately on top of that cell (or 0 if no such cell exists). 5. The <code>down</code> attribute, which is the index of the cell located immediately above that cell (or 0 if no such cell exists). <p>This corresponds to a polyomino [118].</p>
Arc input(s)	CELLS
Arc generator	<i>CLIQUE</i> (\neq) \mapsto collection(cells1, cells2)
Arc arity	2
Arc constraint(s)	$\bigvee \left(\begin{array}{l} \text{cells1.right} = \text{cells2.index} \wedge \text{cells2.left} = \text{cells1.index}, \\ \text{cells1.left} = \text{cells2.index} \wedge \text{cells2.right} = \text{cells1.index}, \\ \text{cells1.up} = \text{cells2.index} \wedge \text{cells2.down} = \text{cells1.index}, \\ \text{cells1.down} = \text{cells2.index} \wedge \text{cells2.up} = \text{cells1.index} \end{array} \right)$

Graph property(ies) • **NVERTEX** = |CELLS|
 • **NCC** = 1

Example

$$\text{polyomino} \left(\left(\begin{array}{ccccc} \text{index} - 1 & \text{right} - 0 & \text{left} - 0 & \text{up} - 2 & \text{down} - 0, \\ \text{index} - 2 & \text{right} - 3 & \text{left} - 0 & \text{up} - 0 & \text{down} - 1, \\ \text{index} - 3 & \text{right} - 0 & \text{left} - 2 & \text{up} - 4 & \text{down} - 0, \\ \text{index} - 4 & \text{right} - 5 & \text{left} - 0 & \text{up} - 0 & \text{down} - 3, \\ \text{index} - 5 & \text{right} - 0 & \text{left} - 4 & \text{up} - 0 & \text{down} - 0 \end{array} \right) \right)$$

Parts (A) and (B) of Figure 4.351 respectively show the initial and final graph. Since we use the **NVERTEX** graph property the vertices of the final graph are stressed in bold. Since we also use the **NCC** graph property we show the unique connected component of the final graph. An arc between two vertices indicates that two cells are directly connected. Figure 4.352 shows the polyomino associated to the previous example.

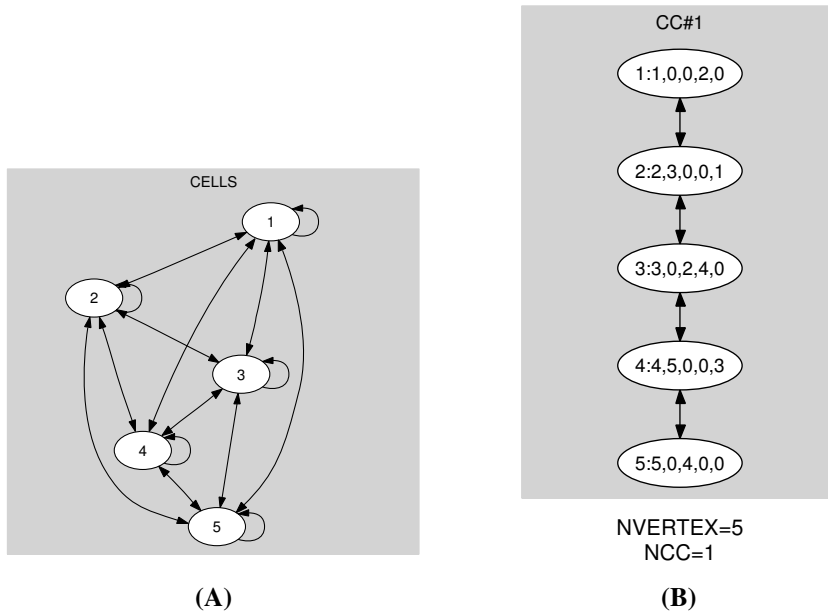


Figure 4.351: Initial and final graph of the polyomino constraint

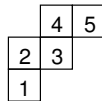


Figure 4.352: Polyomino corresponding to the final graph

Graph model

The graph constraint models the fact that all the cells are connected. We use the *CLIQUE*(≠) arc generator in order to only consider connections between two distinct cells. The first graph property **NVERTEX** = |CELLS| avoid the case isolated cells,

while the second graph property $\mathbf{NCC} = 1$ enforces to have one single group of connected cells.

Signature

From the graph property $\mathbf{NVERTEX} = |\mathbf{CELLS}|$ and from the restriction $|\mathbf{CELLS}| \geq 1$ we have that the final graph is not empty. Therefore it contains at least one connected component. So we can rewrite $\mathbf{NCC} = 1$ to $\mathbf{NCC} \leq 1$ and simplify NCC to NCC.

Usage

Enumeration of polyominoes.

Key words

geometrical constraint, strongly connected component, pentomino.

20000128

747

4.171 product_ctr

Origin	Arithmetic constraint.
Constraint	<code>product_ctr(VARIABLES, CTR, VAR)</code>
Argument(s)	VARIABLES : collection(var – dvar) CTR : atom VAR : dvar
Restriction(s)	required(VARIABLES, var) CTR ∈ [=, ≠, <, ≥, >, ≤]
Purpose	Constraint the product of a set of domain variables. More precisely let P denotes the product of the variables of the VARIABLES collection. Enforce the following constraint to hold: P CTR VAR.
Arc input(s)	VARIABLES
Arc generator	SELF ↦ collection(variables)
Arc arity	1
Arc constraint(s)	TRUE
Graph property(ies)	PRODUCT(VARIABLES, var) CTR VAR
Example	<code>product_ctr({var – 2, var – 1, var – 4}, =, 8)</code>

Parts (A) and (B) of Figure 4.353 respectively show the initial and final graph. Since we use the TRUE arc constraint both graphs are identical.

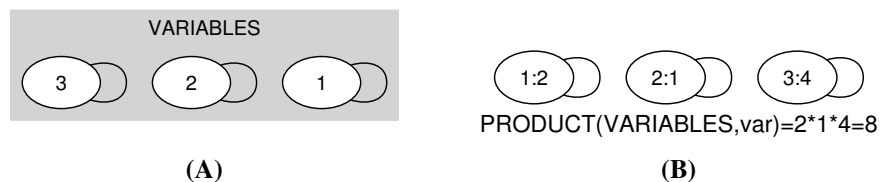


Figure 4.353: Initial and final graph of the `product_ctr` constraint

Graph model	Since we want to keep all the vertices of the initial graph we use the <i>SELF</i> arc generator together with the TRUE arc constraint. This predefined arc constraint always holds.
Used in	<code>cumulative_product</code> .
See also	<code>sum_ctr</code> , <code>range_ctr</code> .
Key words	arithmetic constraint, product.

20030820

749

4.172 range_ctr

Origin	Arithmetic constraint.
Constraint	<code>range_ctr(VARIABLES, CTR, VAR)</code>
Argument(s)	VARIABLES : collection(var – dvar) CTR : atom VAR : dvar
Restriction(s)	required(VARIABLES, var) CTR ∈ [=, ≠, <, ≥, >, ≤]

Purpose

Constraint the difference between the maximum value and the minimum value of a set of domain variables. More precisely let R denotes the difference between the largest and the smallest variables of the `VARIABLES` collection. Enforce the following constraint to hold: $R \text{ CTR } VAR$.

Arc input(s)	VARIABLES
Arc generator	<i>SELF</i> ↦ collection(variables)
Arc arity	1
Arc constraint(s)	TRUE
Graph property(ies)	RANGE (VARIABLES, var) CTR VAR

Example `range_ctr({var – 1, var – 9, var – 4}, =, 8)`

Parts (A) and (B) of Figure 4.354 respectively show the initial and final graph. Since we use the TRUE arc constraint both graphs are identical.

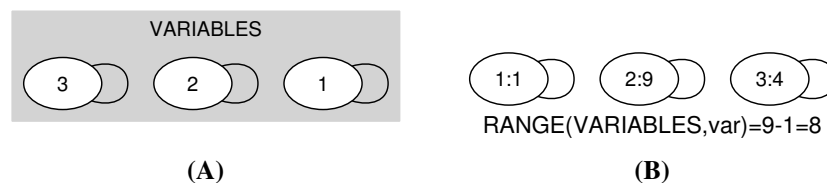


Figure 4.354: Initial and final graph of the `range_ctr` constraint

Graph model	Since we want to keep all the vertices of the initial graph we use the <i>SELF</i> arc generator together with the TRUE arc constraint. This predefined arc constraint always holds.
Used in	<code>shift</code> .
See also	<code>sum_ctr</code> , <code>product_ctr</code> .
Key words	arithmetic constraint, range.

20030820

751

4.173 relaxed_sliding_sum

Origin	CHIP
Constraint	<code>relaxed_sliding_sum(ATLEAST, ATMOST, LOW, UP, SEQ, VARIABLES)</code>
Argument(s)	<pre> ATLEAST : int ATMOST : int LOW : int UP : int SEQ : int VARIABLES : collection(var - dvar) </pre>
Restriction(s)	<pre> ATLEAST ≥ 0 ATMOST ≥ ATLEAST ATMOST ≤ VARIABLES - SEQ + 1 UP ≥ LOW SEQ > 0 SEQ ≤ VARIABLES required(VARIABLES, var) </pre>
Purpose	Constrains that there exist between ATLEAST and ATMOST sequences of SEQ consecutive variables of the collection VARIABLES such that the sum of the variables is in interval [LOW, UP].
Arc input(s)	VARIABLES
Arc generator	<i>PATH</i> \mapsto collection
Arc arity	SEQ
Arc constraint(s)	<ul style="list-style-type: none"> • <code>sum_ctr(collection, ≥, LOW)</code> • <code>sum_ctr(collection, ≤, UP)</code>
Graph property(ies)	<ul style="list-style-type: none"> • $NARC \geq ATLEAST$ • $NARC \leq ATMOST$

Example `relaxed_sliding_sum` $\left(3, 4, 3, 7, 4, \left\{ \begin{array}{l} \text{var} - 2, \\ \text{var} - 4, \\ \text{var} - 2, \\ \text{var} - 0, \\ \text{var} - 0, \\ \text{var} - 3, \\ \text{var} - 4 \end{array} \right\} \right)$

The final directed hypergraph associated to the previous example is given by Figure 4.355. For each vertex of the graph we show its corresponding position within the collection of variables. The constraint associated to each arc corresponds to a conjunction of two `sum_ctr` constraints involving 4 consecutive variables. We did not put vertex

1 since the single arc constraint that mentions vertex 1 does not hold (i.e. the sum $2 + 4 + 2 + 0 = 8$ is not located in interval $[3, 7]$). However, the directed hypergraph contains 3 arcs, so the `relaxed_sliding_sum` constraint is satisfied since it was requested to have between 3 and 4 arcs.

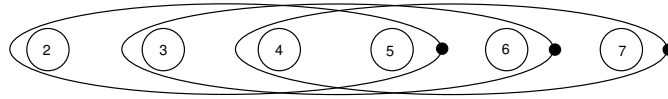


Figure 4.355: Final directed hypergraph associated to the example

Algorithm [65].

See also `sliding_sum`, `sum_ctr`.

Key words sliding sequence constraint, soft constraint, relaxation, sequence, hypergraph.

4.174 same

Origin	N. Beldiceanu
Constraint	$\text{same}(\text{VARIABLES1}, \text{VARIABLES2})$
Argument(s)	VARIABLES1 : collection(var – dvar) VARIABLES2 : collection(var – dvar)
Restriction(s)	$ \text{VARIABLES1} = \text{VARIABLES2} $ required(VARIABLES1, var) required(VARIABLES2, var)
Purpose	The variables of the VARIABLES2 collection correspond to the variables of the VARIABLES1 collection according to a permutation.
Arc input(s)	VARIABLES1 VARIABLES2
Arc generator	$\text{PRODUCT} \mapsto \text{collection}(\text{variables1}, \text{variables2})$
Arc arity	2
Arc constraint(s)	$\text{variables1.var} = \text{variables2.var}$
Graph property(ies)	<ul style="list-style-type: none"> • for all connected components: $\text{NSOURCE} = \text{NSINK}$ • $\text{NSOURCE} = \text{VARIABLES1}$ • $\text{NSINK} = \text{VARIABLES2}$

Example

$$\text{same} \left(\left(\begin{array}{c} \left\{ \begin{array}{c} \text{var} - 1, \\ \text{var} - 9, \\ \text{var} - 1, \\ \text{var} - 5, \\ \text{var} - 2, \\ \text{var} - 1 \end{array} \right\}, \\ \left\{ \begin{array}{c} \text{var} - 9, \\ \text{var} - 1, \\ \text{var} - 1, \\ \text{var} - 1, \\ \text{var} - 2, \\ \text{var} - 5 \end{array} \right\} \end{array} \right) \right)$$

Parts (A) and (B) of Figure 4.356 respectively show the initial and final graph. Since we use the **NSOURCE** and **NSINK** graph properties, the source and sink vertices of the final graph are stressed with a double circle. Since there is a constraint on each connected component of the final graph we also show the different connected components. Each of them corresponds to an equivalence class according to the arc constraint. The **same** constraint holds since:

- Each connected component of the final graph has the same number of sources and of sinks.
- The number of sources of the final graph is equal to $|\text{VARIABLES1}|$.
- The number of sinks of the final graph is equal to $|\text{VARIABLES2}|$.

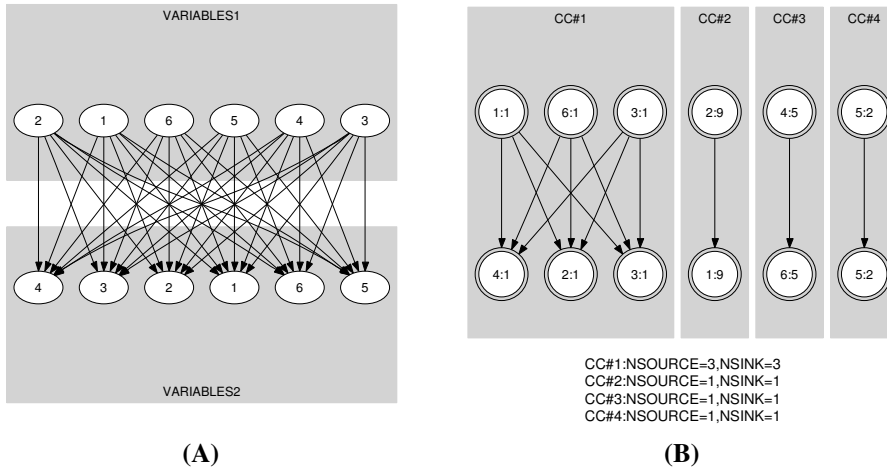


Figure 4.356: Initial and final graph of the same constraint

Signature

Since the initial graph contains only sources and sinks, and since isolated vertices are eliminated from the final graph, we make the following observations:

- Sources of the initial graph cannot become sinks of the final graph,
- Sinks of the initial graph cannot become sources of the final graph.

From the previous observations and since we use the *PRODUCT* arc generator on the collections *VARIABLES1* and *VARIABLES2*, we have that the maximum number of sources and sinks of the final graph is respectively equal to $|\text{VARIABLES1}|$ and $|\text{VARIABLES2}|$. Therefore we can rewrite $\text{NSOURCE} = |\text{VARIABLES1}|$ to $\text{NSOURCE} \geq |\text{VARIABLES1}|$ and simplify $\overline{\text{NSOURCE}}$ to NSOURCE . In a similar way, we can rewrite $\text{NSINK} = |\text{VARIABLES2}|$ to $\text{NSINK} \geq |\text{VARIABLES2}|$ and simplify $\overline{\text{NSINK}}$ to NSINK .

Automaton

To each item of the collection *VARIABLES1* corresponds a signature variable S_i , which is equal to 0. To each item of the collection *VARIABLES2* corresponds a signature variable $S_{i+|\text{VARIABLES1}|}$, which is equal to 1.

Usage

The same constraint can be used in the following contexts:

- Pairing problems taken from [25]. The organization Doctors Without Borders has a list of doctors and a list of nurses, each of whom volunteered to go on one mission in the next year. Each volunteer specifies a list of possible dates and each mission involves one doctor and one nurse. The task is to produce a list of pairs such that

each pair includes a doctor and a nurse who are available at the same date and each volunteer appears in exactly one pair. The problem is modelled by a `same` ($D = d_1, d_2, \dots, d_m, N = n_1, n_2, \dots, n_m$) constraint where each doctor is represented by a domain variable in D and each nurse by a domain variable in N . For a given doctor or nurse the corresponding domain variable gives the dates when the person is available. When the number of nurses is different from the number of doctors we replace the `same` constraint by a `used_by` constraint.

- Timetabling problems where we wish to produce fair schedules for different persons is a second use of the `same` constraint. Assume we need to generate a plan over a period of D consecutive days for P persons. For each day d and each person p we need to decide whether person p works in the morning shift, in the afternoon shift, in the night shift or does not work at all on day d . In a fair schedule, the number of morning shifts should be the same for all the persons. The same condition holds for the afternoon and the night shifts as well as for the days off. We create for each person p the sequence of variables $v_{p,1}, v_{p,2}, \dots, v_{p,D}$. $v_{p,D}$ is equal to one of 0, 1, 2 and 3, depending on whether person p does not work, works in the morning, in the afternoon or during the night on day d . We can use $P - 1$ `same` constraints to express the fact that $v_{1,1}, v_{1,2}, \dots, v_{1,D}$ should be a permutation of $v_{p,1}, v_{p,2}, \dots, v_{p,D}$ for each ($1 < p \leq P$).
- The `same` constraint can also be used as a channelling constraint for modelling the following recurring pattern: Given the number of 1s in each line and each column of a 0-1 matrix \mathcal{M} with n lines and m columns, reconstruct the matrix. This pattern usually occurs with additional constraints about compatible positions of the 1s, or about the overall shape reconstructed from all the 1's (e.g. convexity, connectivity). If we restrict ourself to the basic pattern there is an $O(mn)$ algorithm for reconstructing a $m \cdot n$ matrix from its horizontal and vertical directions [138]. We show how to model this pattern with the `same` constraint. Let l_i ($1 \leq i \leq n$) and c_j ($1 \leq j \leq m$) denote respectively, the required number of 1s in the i th line and the j th column of \mathcal{M} . We number the entries of the matrix as shown in the left-hand side of 4.358. For line i we create l_i domain variables v_{ik} where $k \in [1, l_i]$. Similarly, for each column j we create c_j domain variables u_{jk} where $k \in [1, c_j]$. The domain of each variable contains the set of entries that belong to the row or column that the variable corresponds to. Thus, each domain variable represents a 1 which appears in the designated row or column. Let \mathcal{V} be the set of variables corresponding to rows and \mathcal{U} be the set of variables corresponding to columns. To make sure that each 1 is placed in a different entry, we impose the constraint `alldifferent`(\mathcal{U}). In addition, the constraint `same`(\mathcal{U}, \mathcal{V}) enforces that the 1s exactly coincide on the lines and the columns. A solution is shown on the right-hand side of 4.358. Note that the `same_and_global_cardinality` constraint allows to model the matrix reconstruction problem without the additional `alldifferent` constraint.

Remark

The `same` constraint is a relaxed version of the `sort` constraint introduced in [139]. We don't enforce the second collection of variables to be sorted in increasing order.

If we interpret the collections `VARIABLES1` and `VARIABLES2` as two multisets variables [140], the `same` constraint can be considered as an equality constraint between two multisets variables.

The `same` constraint can be modeled by two `global_cardinality` constraints. For instance, the `same` constraint

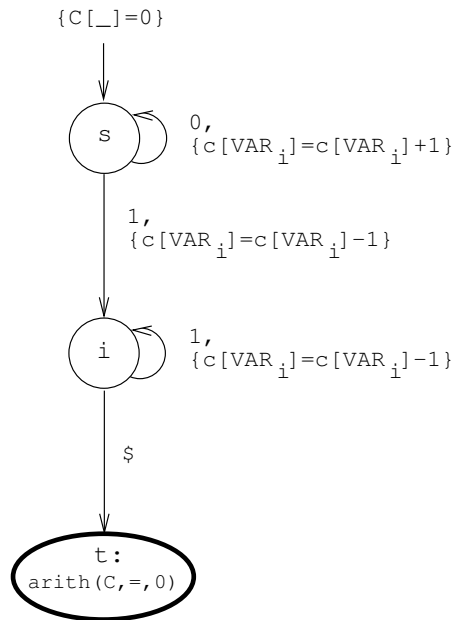


Figure 4.357: Automaton of the same constraint

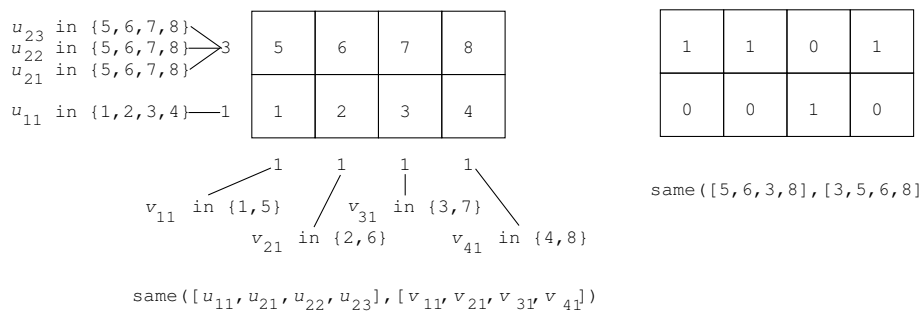


Figure 4.358: Modelling the 0-1 matrix reconstruction problem with the same constraint

$$\text{same} \left(\left\{ \begin{array}{l} \text{var} - x_1, \text{var} - x_2 \\ \text{var} - y_1, \text{var} - y_2 \end{array} \right\}, \right)$$

where the union of the domains of the different variables is $\{1, 2, 3, 4\}$ corresponds to the conjunction of the following two `global_cardinality` constraints:

$$\text{global_cardinality} \left(\left\{ \begin{array}{l} \text{var} - x_1, \text{var} - x_2 \\ \text{val} - 1 \quad \text{noccurrence} - c_1, \\ \text{val} - 2 \quad \text{noccurrence} - c_2, \\ \text{val} - 3 \quad \text{noccurrence} - c_3, \\ \text{val} - 4 \quad \text{noccurrence} - c_4 \end{array} \right\} \right)$$

$$\text{global_cardinality} \left(\left\{ \begin{array}{l} \text{var} - y_1, \text{var} - y_2 \\ \text{val} - 1 \quad \text{noccurrence} - c_1, \\ \text{val} - 2 \quad \text{noccurrence} - c_2, \\ \text{val} - 3 \quad \text{noccurrence} - c_3, \\ \text{val} - 4 \quad \text{noccurrence} - c_4 \end{array} \right\} \right)$$

As shown by the next example, the consistency for all variables of the two `global_cardinality` constraints does not implies consistency for the corresponding `same` constraint. This is for instance the case when the domains of x_1 , x_2 , y_1 and y_2 is respectively equal to $\{1, 2\}$, $\{3, 4\}$, $\{1, 2, 3, 4\}$ and $\{3, 4\}$. The conjunction of the two `global_cardinality` constraints does not remove values 3 and 4 from y_1 .

In his PhD thesis, W.-J. van Hoeve introduces a soft version of the `same` constraint where the cost is the minimum number of variables to unassign in order to get back to a solution [104, page 78]. In the context of the `same` constraint this violation cost corresponds to the difference between the number of variables in `VARIABLES1` and the number of values which both occur in `VARIABLES1` and in `VARIABLES2` (provided that one value of `VARIABLES1` matches at most one value of `VARIABLES2`).

Algorithm

In [141], [25] and [142] it is shown how to model this constraint by a flow network that enables to compute arc-consistency and bound-consistency. Unlike the networks used for `alldifferent` and `global_cardinality`, the network now has three sets of nodes, so the algorithms are more complex, in particular the efficient bound-consistency algorithm.

See also

`colored_matrix`, `correspondence`, `same_interval`, `same_modulo`, `same_partition`, `same_and_global_cardinality`, `same_intersection`.

Key words

constraint between two collections of variables, channeling constraint, permutation, multiset, equality between multisets, flow, bound-consistency, automaton, automaton with array of counters.

20000128

759

4.175 same_and_global_cardinality

Origin	Derived from same and global_cardinality
Constraint	same_and_global_cardinality(VARIABLES1, VARIABLES2, VALUES)
Synonym(s)	sgcc, same_gcc, same_and_gcc, swc, same_with_cardinalities.
Argument(s)	VARIABLES1 : collection(var – dvar) VARIABLES2 : collection(var – dvar) VALUES : collection(val – int, noccurrence – dvar)
Restriction(s)	$ VARIABLES1 = VARIABLES2 $ required(VARIABLES1, var) required(VARIABLES2, var) required(VALUES, [val, noccurrence]) distinct(VALUES, val) $VALUES.noccurrence \geq 0$ $VALUES.noccurrence \leq VARIABLES1 $
Purpose	The variables of the VARIABLES2 collection correspond to the variables of the VARIABLES1 collection according to a permutation. In addition, each value $VALUES[i].val$ ($1 \leq i \leq VALUES $) should be taken by exactly $VALUES[i].noccurrence$ variables of the VARIABLES1 collection.
Arc input(s)	VARIABLES1 VARIABLES2
Arc generator	<i>PRODUCT</i> \mapsto collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	variables1.var = variables2.var
Graph property(ies)	<ul style="list-style-type: none"> • for all connected components: NSOURCE = NSINK • NSOURCE = $VARIABLES1$ • NSINK = $VARIABLES2$
	For all items of VALUES:
Arc input(s)	VARIABLES1
Arc generator	<i>SELF</i> \mapsto collection(variables)
Arc arity	1
Arc constraint(s)	variables.var = VALUES.val
Graph property(ies)	NVERTEX = VALUES.noccurrence

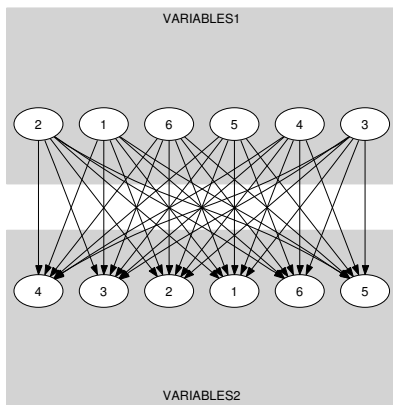
Example

same_and_global_cardinality

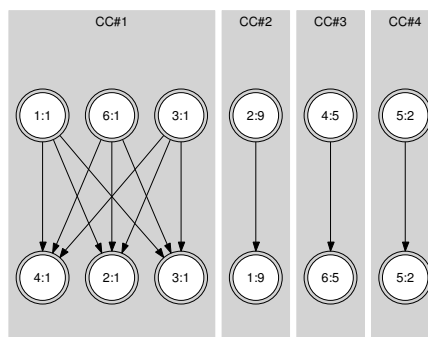
$$\left(\begin{array}{l} \left\{ \begin{array}{l} \text{var} - 1, \\ \text{var} - 9, \\ \text{var} - 1, \\ \text{var} - 5, \\ \text{var} - 2, \\ \text{var} - 1 \end{array} \right\}, \\ \left\{ \begin{array}{l} \text{var} - 9, \\ \text{var} - 1, \\ \text{var} - 1, \\ \text{var} - 1, \\ \text{var} - 2, \\ \text{var} - 5 \end{array} \right\}, \\ \left\{ \begin{array}{l} \text{val} - 1 \quad \text{noccurrence} - 3, \\ \text{val} - 2 \quad \text{noccurrence} - 1, \\ \text{val} - 5 \quad \text{noccurrence} - 1, \\ \text{val} - 7 \quad \text{noccurrence} - 0, \\ \text{val} - 9 \quad \text{noccurrence} - 1 \end{array} \right\} \end{array} \right)$$

Parts (A) and (B) of Figure 4.359 respectively show the initial and final graph associated to the first graph constraint. Since we use the **NSOURCE** and **NSINK** graph properties, the source and sink vertices of the final graph are stressed with a double circle. Since there is a constraint on each connected component of the final graph we also show the different connected components. Each of them corresponds to an equivalence class according to the arc constraint. The `same_and_global_cardinality` constraint holds since:

- The values 1, 9, 1, 5, 2, 1 assigned to `|VARIABLES1|` correspond to a permutation of the values 9, 1, 1, 1, 2, 5 assigned to `|VARIABLES2|`.
- The values 1, 2, 5, 7 and 6 are respectively used 3, 1, 1, 0 and 1 times.



(A)



CC#1: NSOURCE=3, NSINK=3
 CC#2: NSOURCE=1, NSINK=1
 CC#3: NSOURCE=1, NSINK=1
 CC#4: NSOURCE=1, NSINK=1

(B)

Figure 4.359: Initial and final graph of the `same_and_global_cardinality` constraint

Usage	The <code>same_and_global_cardinality</code> constraint can be used for modeling the following assignment problem with one single constraint. The organization Doctors Without Borders has a list of doctors and a list of nurses, each of whom volunteered to go on one rescue mission. Each volunteer specifies a list of possible dates and each mission should include one doctor and one nurse. In addition we have for each date the minimum and maximum number of missions that should be effectively done. The task is to produce a list of pairs such that each pair includes a doctor and a nurse who are available on the same date and each volunteer appears in exactly one pair so that for each day we build the required number of missions.
Algorithm	In [143], the flow network that was used to model the <code>same</code> constraint [141, 25] is extended to support the cardinalities. Then, algorithms are developed to compute arc-consistency and bound-consistency.
See also	<code>same</code> , <code>global_cardinality</code> .
Key words	constraint between two collections of variables, value constraint, permutation, multiset, equality between multisets, assignment, demand profile.

20040530

763

4.176 same_intersection

Origin	Derived from same and common.
Constraint	<code>same_intersection(VARIABLES1, VARIABLES2)</code>
Argument(s)	VARIABLES1 : collection(var – dvar) VARIABLES2 : collection(var – dvar)
Restriction(s)	required(VARIABLES1, var) required(VARIABLES2, var)
Purpose	Each value which occurs both in the VARIABLES1 and in the VARIABLES2 collections has the same number of occurrences in VARIABLES1 as well as in VARIABLES2.
Arc input(s)	VARIABLES1 VARIABLES2
Arc generator	$PRODUCT \mapsto \text{collection}(\text{variables1}, \text{variables2})$
Arc arity	2
Arc constraint(s)	<code>variables1.var = variables2.var</code>
Graph property(ies)	for all connected components: $NSOURCE = NSINK$

Example	$\text{same_intersection} \left(\left(\begin{array}{c} \{ \text{var} - 1, \\ \text{var} - 9, \\ \text{var} - 1, \\ \text{var} - 5, \\ \text{var} - 2, \\ \text{var} - 1 \} \\ \{ \text{var} - 9, \\ \text{var} - 1, \\ \text{var} - 1, \\ \text{var} - 1, \\ \text{var} - 3, \\ \text{var} - 5, \\ \text{var} - 8 \} \end{array} \right) \right)$
----------------	--

Parts (A) and (B) of Figure 4.360 respectively show the initial and final graph. The `same_intersection` constraint holds since each connected component of the final graph has the same number of sources and sinks. Note that all the vertices corresponding to the variables that take values 2, 3 or 8 were removed from the final graph since there is no arc for which the associated equality constraint holds.

See also `same`, `common`, `alldifferent_on_intersection`, `nvalue_on_intersection`.

Key words constraint between two collections of variables, constraint on the intersection.

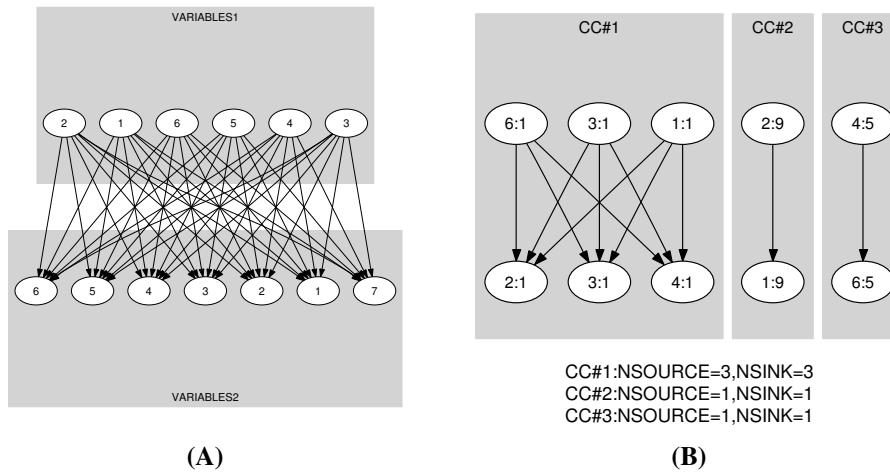


Figure 4.360: Initial and final graph of the same_intersection constraint

4.177 same_interval

Origin	Derived from same.
Constraint	<code>same_interval(VARIABLES1, VARIABLES2, SIZE_INTERVAL)</code>
Argument(s)	<code>VARIABLES1</code> : collection(var – dvar) <code>VARIABLES2</code> : collection(var – dvar) <code>SIZE_INTERVAL</code> : int
Restriction(s)	<code> VARIABLES1 = VARIABLES2 </code> <code>required(VARIABLES1, var)</code> <code>required(VARIABLES2, var)</code> <code>SIZE_INTERVAL > 0</code>
Purpose	Let N_i (respectively M_i) denote the number of variables of the collection <code>VARIABLES1</code> (respectively <code>VARIABLES2</code>) that take a value in the interval <code>[SIZE_INTERVAL · i, SIZE_INTERVAL · i + SIZE_INTERVAL – 1]</code> . For all integer i we have $N_i = M_i$.
Arc input(s)	<code>VARIABLES1 VARIABLES2</code>
Arc generator	<code>PRODUCT</code> \mapsto <code>collection(variables1, variables2)</code>
Arc arity	2
Arc constraint(s)	<code>variables1.var/SIZE_INTERVAL = variables2.var/SIZE_INTERVAL</code>
Graph property(ies)	<ul style="list-style-type: none"> • for all connected components: <code>NSOURCE = NSINK</code> • <code>NSOURCE = VARIABLES1 </code> • <code>NSINK = VARIABLES2 </code>

Example

$$\text{same_interval} \left(\left(\left\{ \begin{array}{l} \text{var} - 1, \\ \text{var} - 7, \\ \text{var} - 6, \\ \text{var} - 0, \\ \text{var} - 1, \\ \text{var} - 7 \end{array} \right\}, \right. \right. \\ \left. \left. \left\{ \begin{array}{l} \text{var} - 8, \\ \text{var} - 8, \\ \text{var} - 8, \\ \text{var} - 0, \\ \text{var} - 1, \\ \text{var} - 2 \end{array} \right\}, 3 \right) \right)$$

In the previous example, the third parameter `SIZE_INTERVAL` defines the following family of intervals $[3 \cdot k, 3 \cdot k + 2]$, where k is an integer. Parts (A) and (B) of Figure 4.361 respectively show the initial and final graph. Since we use the `NSOURCE` and `NSINK` graph properties, the source and sink vertices of the final graph are stressed with a double circle. Since there is a constraint on each connected component of the final

graph we also show the different connected components. Each of them corresponds to an equivalence class according to the arc constraint. The `same_interval` constraint holds since:

- Each connected component of the final graph has the same number of sources and of sinks.
- The number of sources of the final graph is equal to $|\text{VARIABLES1}|$.
- The number of sinks of the final graph is equal to $|\text{VARIABLES2}|$.

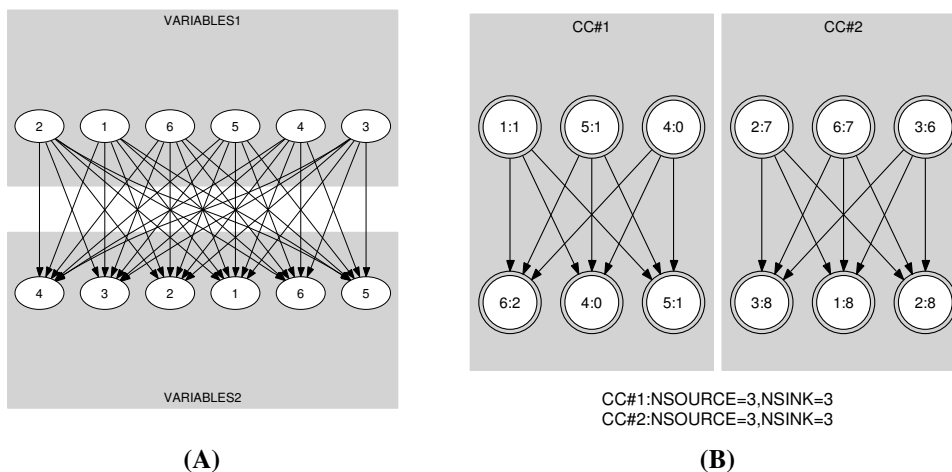


Figure 4.361: Initial and final graph of the `same_interval` constraint

Signature

Since the initial graph contains only sources and sinks, and since isolated vertices are eliminated from the final graph, we make the following observations:

- Sources of the initial graph cannot become sinks of the final graph,
- Sinks of the initial graph cannot become sources of the final graph.

From the previous observations and since we use the *PRODUCT* arc generator on the collections `VARIABLES1` and `VARIABLES2`, we have that the maximum number of sources and sinks of the final graph is respectively equal to $|\text{VARIABLES1}|$ and $|\text{VARIABLES2}|$. Therefore we can rewrite $\text{NSOURCE} = |\text{VARIABLES1}|$ to $\text{NSOURCE} \geq |\text{VARIABLES1}|$ and simplify $\underline{\text{NSOURCE}}$ to $\overline{\text{NSOURCE}}$. In a similar way, we can rewrite $\text{NSINK} = |\text{VARIABLES2}|$ to $\text{NSINK} \geq |\text{VARIABLES2}|$ and simplify $\underline{\text{NSINK}}$ to $\overline{\text{NSINK}}$.

Algorithm

See algorithm of the `same` constraint.

See also

`same`.

Key words

constraint between two collections of variables, permutation, interval.

4.178 same_modulo

Origin	Derived from same.
Constraint	<code>same_modulo(VARIABLES1, VARIABLES2, M)</code>
Argument(s)	<p>VARIABLES1 : collection(var – dvar) VARIABLES2 : collection(var – dvar) M : int</p>
Restriction(s)	<p>$\text{VARIABLES1} = \text{VARIABLES2}$ required(VARIABLES1, var) required(VARIABLES2, var) $M > 0$</p>
Purpose	<p>For each integer R in $[0, M - 1]$, let $N1_R$ (respectively $N2_R$) denote the number of variables of VARIABLE1 (respectively VARIABLE2) which have R as a rest when divided by M. For all R in $[0, M - 1]$ we have that $N1_R = N2_R$.</p>
Arc input(s)	VARIABLES1 VARIABLES2
Arc generator	$\text{PRODUCT} \mapsto \text{collection}(\text{variables1}, \text{variables2})$
Arc arity	2
Arc constraint(s)	$\text{variables1.var mod } M = \text{variables2.var mod } M$
Graph property(ies)	<ul style="list-style-type: none"> • for all connected components: $\text{NSOURCE} = \text{NSINK}$ • $\text{NSOURCE} = \text{VARIABLES1}$ • $\text{NSINK} = \text{VARIABLES2}$

Example	$\text{same_modulo} \left(\left(\left\{ \begin{array}{l} \text{var} - 1, \\ \text{var} - 9, \\ \text{var} - 1, \\ \text{var} - 5, \\ \text{var} - 2, \\ \text{var} - 1 \end{array} \right\}, \right. \right. \\ \left. \left. \left\{ \begin{array}{l} \text{var} - 6, \\ \text{var} - 4, \\ \text{var} - 1, \\ \text{var} - 1, \\ \text{var} - 5, \\ \text{var} - 5 \end{array} \right\}, 3 \right)$
----------------	--

Parts (A) and (B) of Figure 4.362 respectively show the initial and final graph. Since we use the **NSOURCE** and **NSINK** graph properties, the source and sink vertices of the final graph are stressed with a double circle. Since there is a constraint on each connected component of the final graph we also show the different connected components. Each of them corresponds to an equivalence class according to the arc constraint. The `same_modulo` constraint holds since:

- Each connected component of the final graph has the same number of sources and of sinks.
- The number of sources of the final graph is equal to $|\text{VARIABLES1}|$.
- The number of sinks of the final graph is equal to $|\text{VARIABLES2}|$.

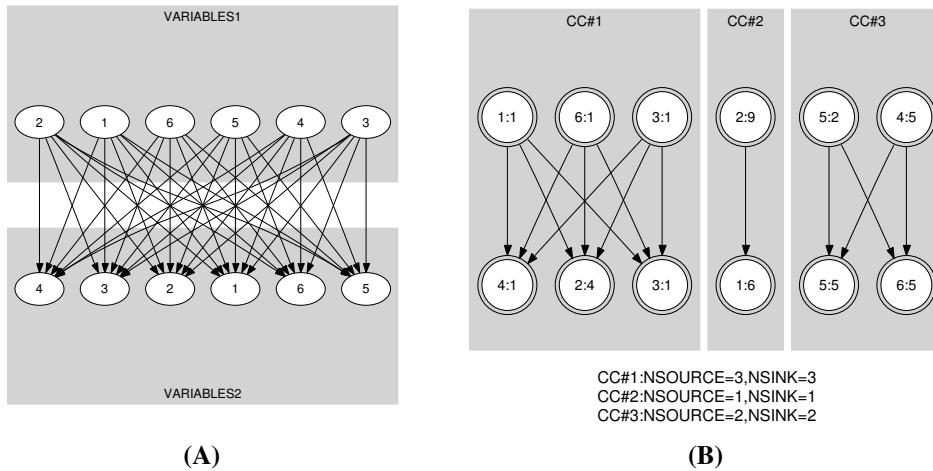


Figure 4.362: Initial and final graph of the `same_modulo` constraint

Signature

Since the initial graph contains only sources and sinks, and since isolated vertices are eliminated from the final graph, we make the following observations:

- Sources of the initial graph cannot become sinks of the final graph,
- Sinks of the initial graph cannot become sources of the final graph.

From the previous observations and since we use the *PRODUCT* arc generator on the collections `VARIABLES1` and `VARIABLES2`, we have that the maximum number of sources and sinks of the final graph is respectively equal to $|\text{VARIABLES1}|$ and $|\text{VARIABLES2}|$. Therefore we can rewrite $\text{NSOURCE} = |\text{VARIABLES1}|$ to $\text{NSOURCE} \geq |\text{VARIABLES1}|$ and simplify $\overline{\text{NSOURCE}}$ to NSOURCE . In a similar way, we can rewrite $\text{NSINK} = |\text{VARIABLES2}|$ to $\text{NSINK} \geq |\text{VARIABLES2}|$ and simplify $\overline{\text{NSINK}}$ to NSINK .

See also

`same`.

Key words

constraint between two collections of variables, permutation, modulo.

4.179 same_partition

Origin	Derived from same.
Constraint	same_partition(VARIABLES1, VARIABLES2, PARTITIONS)
Type(s)	VALUES : collection(val – int)
Argument(s)	VARIABLES1 : collection(var – dvar) VARIABLES2 : collection(var – dvar) PARTITIONS : collection(p – VALUES)
Restriction(s)	required(VALUES, val) distinct(VALUES, val) $ VARIABLES1 = VARIABLES2 $ required(VARIABLES1, var) required(VARIABLES2, var) required(PARTITIONS, p) $ PARTITIONS \geq 2$
Purpose	For each integer i in $[1, PARTITIONS]$, let $N1_i$ (respectively $N2_i$) denote the number of variables of VARIABLES1 (respectively VARIABLES2) which take their value in the i^{th} partition of the collection PARTITIONS. For all i in $[1, PARTITIONS]$ we have $N1_i = N2_i$.
Arc input(s)	VARIABLES1 VARIABLES2
Arc generator	<i>PRODUCT</i> \mapsto collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	in_same_partition(variables1.var, variables2.var, PARTITIONS)
Graph property(ies)	<ul style="list-style-type: none"> • for all connected components: $NSOURCE = NSINK$ • $NSOURCE = VARIABLES1$ • $NSINK = VARIABLES2$

Example

$$\text{same} \left(\left(\left\{ \begin{array}{l} \text{var} - 1, \\ \text{var} - 2, \\ \text{var} - 6, \\ \text{var} - 3, \\ \text{var} - 1, \\ \text{var} - 2 \end{array} \right\}, \left\{ \begin{array}{l} \text{var} - 6, \\ \text{var} - 6, \\ \text{var} - 2, \\ \text{var} - 3, \\ \text{var} - 1, \\ \text{var} - 3 \end{array} \right\}, \left\{ \begin{array}{l} p - \{\text{val} - 1, \text{val} - 3\}, \\ p - \{\text{val} - 4\}, \\ p - \{\text{val} - 2, \text{val} - 6\} \end{array} \right\} \right) \right)$$

Parts (A) and (B) of Figure 4.363 respectively show the initial and final graph. Since we use the **NSOURCE** and **NSINK** graph properties, the source and sink vertices of the final graph are stressed with a double circle. Since there is a constraint on each connected component of the final graph we also show the different connected components. Each of them corresponds to an equivalence class according to the arc constraint. The `same_partition` constraint holds since:

- Each connected component of the final graph has the same number of sources and of sinks.
- The number of sources of the final graph is equal to $|\text{VARIABLES1}|$.
- The number of sinks of the final graph is equal to $|\text{VARIABLES2}|$.

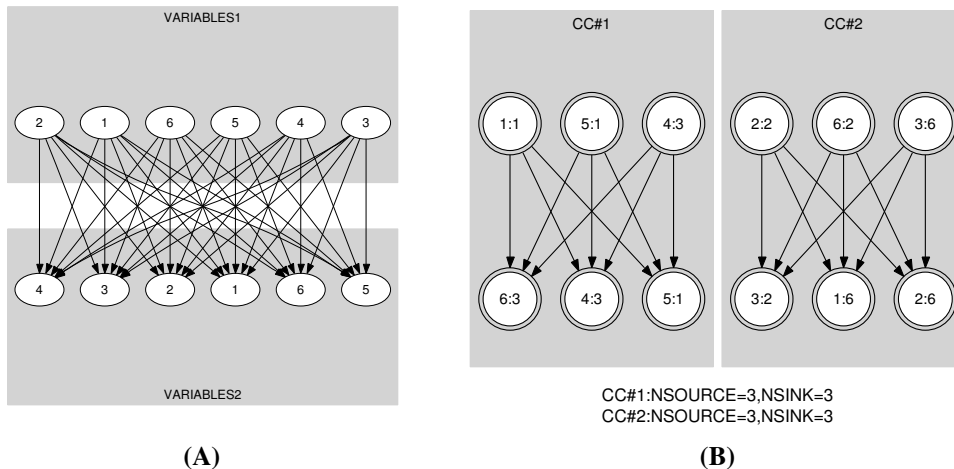


Figure 4.363: Initial and final graph of the `same_partition` constraint

Signature

Since the initial graph contains only sources and sinks, and since isolated vertices are eliminated from the final graph, we make the following observations:

- Sources of the initial graph cannot become sinks of the final graph,
- Sinks of the initial graph cannot become sources of the final graph.

From the previous observations and since we use the *PRODUCT* arc generator on the collections `VARIABLES1` and `VARIABLES2`, we have that the maximum number of sources and sinks of the final graph is respectively equal to $|\text{VARIABLES1}|$ and $|\text{VARIABLES2}|$. Therefore we can rewrite $\text{NSOURCE} = |\text{VARIABLES1}|$ to $\text{NSOURCE} \geq |\text{VARIABLES1}|$ and simplify $\underline{\text{NSOURCE}}$ to $\overline{\text{NSOURCE}}$. In a similar way, we can rewrite $\text{NSINK} = |\text{VARIABLES2}|$ to $\text{NSINK} \geq |\text{VARIABLES2}|$ and simplify $\underline{\text{NSINK}}$ to $\overline{\text{NSINK}}$.

See also

`same, in_same_partition`.

Key words

constraint between two collections of variables, permutation, partition.

4.180 sequence_folding

Origin	J. Pearson
Constraint	<code>sequence_folding(LETTERS)</code>
Argument(s)	<code>LETTERS : collection(index - int, next - dvar)</code>
Restriction(s)	$ \text{LETTERS} \geq 1$ <code>required(LETTERS, [index, next])</code> $\text{LETTERS.index} \geq 1$ $\text{LETTERS.index} \leq \text{LETTERS} $ <code>increasing_seq(LETTERS, index)</code> $\text{LETTERS.next} \geq 1$ $\text{LETTERS.next} \leq \text{LETTERS} $
Purpose	Express the fact that a sequence is folded in a way that no crossing occurs. A sequence is modelled by a collection of letters. For each letter l_1 of a sequence, we indicate the next letter l_2 located after l_1 which is directly in contact with l_1 (l_1 itself if such a letter does not exist).
Arc input(s)	LETTERS
Arc generator	$\text{SELF} \mapsto \text{collection}(\text{letters})$
Arc arity	1
Arc constraint(s)	$\text{letters.next} \geq \text{letters.index}$
Graph property(ies)	$\text{NARC} = \text{LETTERS} $
Arc input(s)	LETTERS
Arc generator	$\text{CLIQUE}(<) \mapsto \text{collection}(\text{letters1}, \text{letters2})$
Arc arity	2
Arc constraint(s)	$\text{letters2.index} \geq \text{letters1.next} \vee \text{letters2.next} \leq \text{letters1.next}$
Graph property(ies)	$\text{NARC} = \text{LETTERS} * (\text{LETTERS} - 1) / 2$
Example	$\text{sequence_folding} \left(\left(\begin{array}{l} \text{index} - 1 \quad \text{next} - 1, \\ \text{index} - 2 \quad \text{next} - 8, \\ \text{index} - 3 \quad \text{next} - 3, \\ \text{index} - 4 \quad \text{next} - 5, \\ \text{index} - 5 \quad \text{next} - 5, \\ \text{index} - 6 \quad \text{next} - 7, \\ \text{index} - 7 \quad \text{next} - 7, \\ \text{index} - 8 \quad \text{next} - 8, \\ \text{index} - 9 \quad \text{next} - 9 \end{array} \right) \right)$

Parts (A) and (B) of Figure 4.364 respectively show the initial and final graph. Since we use the **NARC** graph property, the arcs of the final graph are stressed in bold. Figure 4.365 gives the folded sequence associated to the previous example. Each number represents the index of an item.

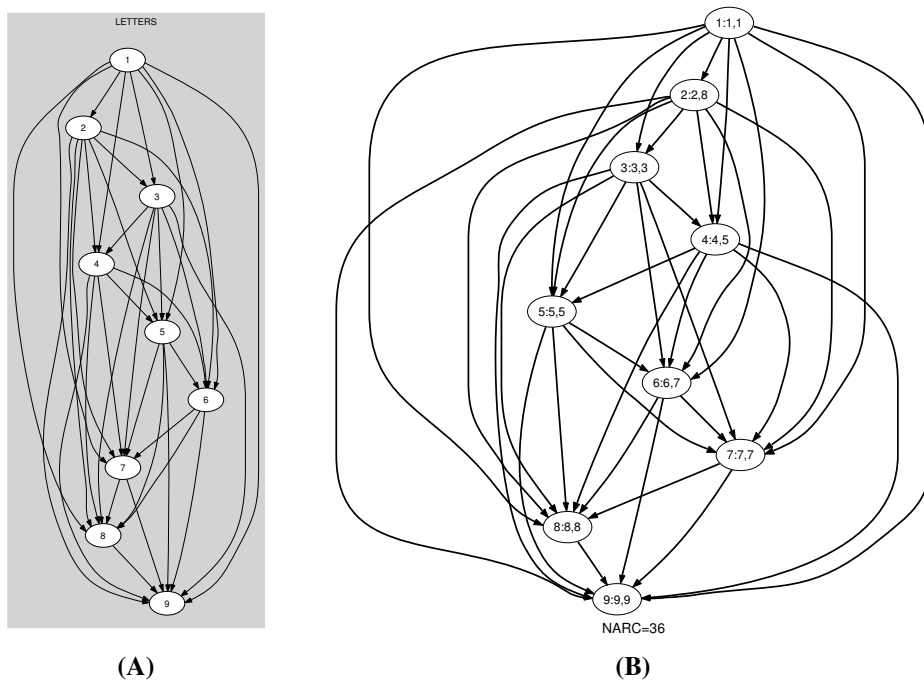


Figure 4.364: Initial and final graph of the `sequence_folding` constraint

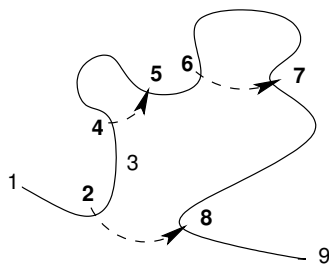


Figure 4.365: Folded sequence associated to the example

Graph model

In the list of restrictions note the `increasing` statement which imposes the items of the `LETTERS` collection to be ordered in increasing order of their index attribute. This is used so that the arc generator `CLIQUE(<)` only generates arcs between vertices for which the indices are increasing. The arc constraint of the second graph constraint avoids the following conditions to be both true:

- The second letter is located before the letter associated to the first letter,
- The letter associated to the second letter is located after the letter associated to the first letter.

Observe that, from the previous remark, we know that the first letter is located before the second letter. The graph property enforces all arcs constraints to hold.

Signature

Consider the first graph constraint. Since we use the *SELF* arc generator on the *LETTERS* collection the maximum number of arcs of the final graph is equal to $|\text{LETTERS}|$. Therefore we can rewrite the graph property $\text{NARC} = |\text{LETTERS}|$ to $\text{NARC} \geq |\text{LETTERS}|$ and simplify $\overline{\text{NARC}}$ to $\overline{\text{NARC}}$.

Consider now the second graph constraint. Since we use the *CLIQUE*($<$) arc generator on the *LETTERS* collection the maximum number of arcs of the final graph is equal to $|\text{LETTERS}| \cdot (|\text{LETTERS}| - 1)/2$. Therefore we can rewrite the graph property $\text{NARC} = |\text{LETTERS}| \cdot (|\text{LETTERS}| - 1)/2$ to $\text{NARC} \geq |\text{LETTERS}| \cdot (|\text{LETTERS}| - 1)/2$ and simplify $\overline{\text{NARC}}$ to $\overline{\text{NARC}}$.

Automaton

Figure 4.366 depicts the automaton associated to the *sequence_folding* constraint. Consider the i^{th} and the j^{th} ($i < j$) items of the collection *LETTERS*. Let INDEX_i and NEXT_i respectively denote the *index* and the *next* attributes of the i^{th} item of the collection *LETTERS*. Similarly, let INDEX_j and NEXT_j respectively denote the *index* and the *next* attributes of the j^{th} item of the collection *LETTERS*. To each quadruple $(\text{INDEX}_i, \text{NEXT}_i, \text{INDEX}_j, \text{NEXT}_j)$ corresponds a signature variable $S_{i,j}$, which takes its value in $\{0, 1, 2\}$, as well as the following signature constraint:

$$(\text{INDEX}_i \leq \text{NEXT}_i) \wedge (\text{INDEX}_j \leq \text{NEXT}_j) \wedge (\text{NEXT}_i \leq \text{NEXT}_j) \Leftrightarrow S_{i,j} = 0 \wedge$$

$$(\text{INDEX}_i \leq \text{NEXT}_i) \wedge (\text{INDEX}_j \leq \text{NEXT}_j) \wedge (\text{NEXT}_i > \text{INDEX}_j) \wedge (\text{NEXT}_j \leq \text{NEXT}_i) \Leftrightarrow S_{i,j} = 1.$$

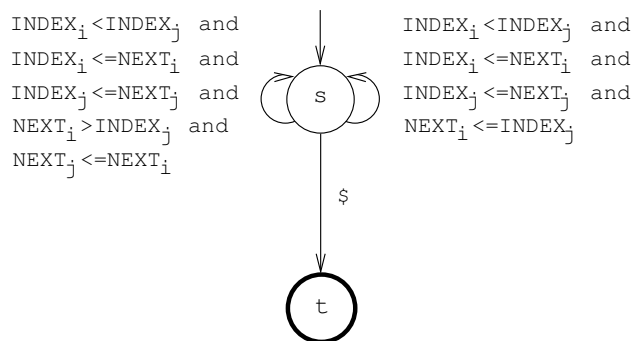


Figure 4.366: Automaton of the *sequence_folding* constraint

Usage

Motivated by RNA folding [144].

Key words

decomposition, geometrical constraint, sequence, bioinformatics, automaton, automaton without counters.

20030820

775

4.181 set_value_precede

Origin [121]

Constraint `set_value_precede(S, T, VARIABLES)`

Argument(s)
 S : int
 T : int
 VARIABLES : collection(var – svar)

Restriction(s)
 $S \neq T$
 required(VARIABLES, var)

Purpose

If there exists a set variable v_1 of VARIABLES such that S does not belong to v_1 and T does, then there also exists a set variable v_2 preceding v_1 such that S belongs to v_2 and T does not.

Example

$$\text{set_value_precede} \left(2, 1, \left\{ \begin{array}{l} \text{var} - \{0, 2\}, \\ \text{var} - \{0, 1\}, \\ \text{var} - \emptyset, \\ \text{var} - \{1\} \end{array} \right\} \right)$$

The `set_value_precede` constraint holds since the first occurrence of value 2 precedes the first occurrence of value 1.

Algorithm

A filtering algorithm for maintaining value precedence on a sequence of set variables is presented in [121]. Its complexity is linear to the number of variables of the collection VARIABLES.

See also `int_value_precede`.

Key words order constraint, symmetry, indistinguishable values, value precedence, constraint involving set variables.

20041003

777

4.182 shift

Origin	N. Beldiceanu
Constraint	<code>shift(MIN_BREAK, MAX_RANGE, TASKS)</code>
Argument(s)	<code>MIN_BREAK : int</code> <code>MAX_RANGE : int</code> <code>TASKS : collection(id - int, origin - dvar, end - dvar)</code>
Restriction(s)	<code>MIN_BREAK > 0</code> <code>MAX_RANGE > 0</code> <code>required(TASKS, [id, origin, end])</code> <code>distinct(TASKS, id)</code>
Purpose	<div style="border: 1px solid black; padding: 5px;"> <p>The difference between the end of the last task of a <i>shift</i> and the origin of the first task of a <i>shift</i> should not exceed the quantity <code>MAX_RANGE</code>. Two tasks t_1 and t_2 belong to the <i>same shift</i> if at least one of the following conditions is true:</p> <ul style="list-style-type: none"> • Task t_2 starts after the end of task t_1 at a distance that is less than or equal to the quantity <code>MIN_BREAK</code>, • Task t_1 starts after the end of task t_2 at a distance that is less than or equal to the quantity <code>MIN_BREAK</code>. • Task t_1 overlaps task t_2. </div>
Arc input(s)	<code>TASKS</code>
Arc generator	<code>SELF</code> \mapsto <code>collection(tasks)</code>
Arc arity	1
Arc constraint(s)	<ul style="list-style-type: none"> • <code>tasks.end \geq tasks.origin</code> • <code>tasks.end - tasks.origin \leq MAX_RANGE</code>
Graph property(ies)	<code>NARC = TASKS </code>
Arc input(s)	<code>TASKS</code>
Arc generator	<code>CLIQUE</code> \mapsto <code>collection(tasks1, tasks2)</code>
Arc arity	2
Arc constraint(s)	$\bigvee \left(\begin{array}{l} \text{tasks2.origin} \geq \text{tasks1.end} \wedge \text{tasks2.origin} - \text{tasks1.end} \leq \text{MIN_BREAK}, \\ \text{tasks1.origin} \geq \text{tasks2.end} \wedge \text{tasks1.origin} - \text{tasks2.end} \leq \text{MIN_BREAK}, \\ \text{tasks2.origin} < \text{tasks1.end} \wedge \text{tasks1.origin} < \text{tasks2.end} \end{array} \right)$
Sets	<code>CC</code> \mapsto $\left[\text{variables} - \text{col} \left(\begin{array}{l} \text{VARIABLES} - \text{collection}(\text{var} - \text{dvar}), \\ [\text{item}(\text{var} - \text{TASKS.origin}), \text{item}(\text{var} - \text{TASKS.end})] \end{array} \right) \right]$

Constraint(s) on sets `range_ctr(variables, ≤, MAX_RANGE)`

Example

$$\text{shift} \left(6, 8, \left\{ \begin{array}{l} \text{id} - 1 \quad \text{origin} - 17 \quad \text{end} - 20, \\ \text{id} - 2 \quad \text{origin} - 7 \quad \text{end} - 10, \\ \text{id} - 3 \quad \text{origin} - 2 \quad \text{end} - 4, \\ \text{id} - 4 \quad \text{origin} - 21 \quad \text{end} - 22, \\ \text{id} - 5 \quad \text{origin} - 5 \quad \text{end} - 6 \end{array} \right\} \right)$$

Parts (A) and (B) of Figure 4.367 respectively show the initial and final graph associated to the second graph constraint. Since we use the set generator CC we show the two connected components of the final graph. They respectively correspond to the two shifts which are displayed in Figure 4.368. Each task is drawn as a rectangle with its corresponding id in the middle. We indicate the distance between two consecutive tasks of a same shift and check that it is less than or equal to the value of the MIN_BREAK parameter (6 in the example). Since each shift has a range that is less than or equal to the MAX_RANGE parameter, the shift constraint holds (the range of a shift is the difference between the end of the last task of the shift and the origin of the first task of the shift).

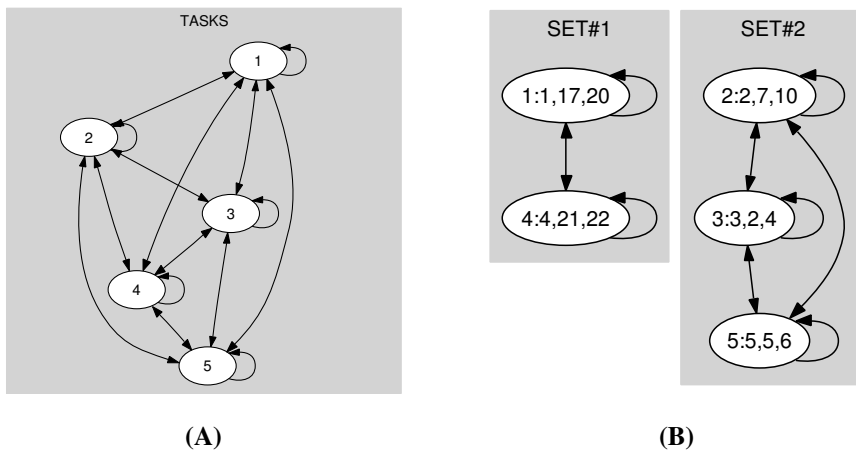


Figure 4.367: Initial and final graph of the shift constraint

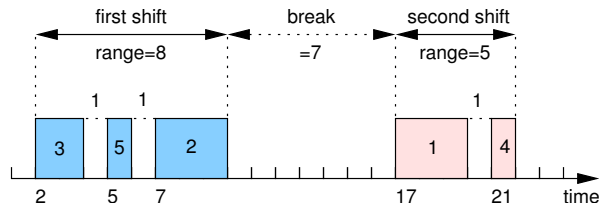


Figure 4.368: The two shifts of the example

Graph model

The first graph constraint enforces the following two constraints between the attributes of each task:

- The end of a task should not be situated before its start,
- The duration of a task should not be greater than the *MAX_RANGE* parameter.

The second graph constraint decomposes the final graph in connected components where each component corresponds to a given shift. Finally, the constraint(s) on sets field restricts the stretch of each shift.

Signature

Consider the first graph constraint. Since we use the *SELF* arc generator on the *TASKS* collection the maximum number of arcs of the final graph is equal to $|\text{TASKS}|$. Therefore we can rewrite the graph property $\text{NARC} = |\text{TASKS}|$ to $\text{NARC} \geq |\text{TASKS}|$ and simplify $\overline{\text{NARC}}$ to $\overline{\text{NARC}}$.

Usage

The shift constraint can be used in machine scheduling problems where one has to shut down a machine for maintenance purpose after a given maximum utilisation of that machine. In this case the *MAX_RANGE* parameter indicates the maximum possible utilisation of the machine before maintenance, while the *MIN_BREAK* parameter gives the minimum time needed for maintenance.

The shift constraint can also be used for timetabling problems where the rest period of a person can move in time. In this case *MAX_RANGE* indicates the maximum possible working time for a person, while *MIN_BREAK* specifies the minimum length of the break that follows a working time period.

See also

sliding_time_window.

Key words

scheduling constraint, timetabling constraint, temporal constraint.

20030820

781

4.183 `size_maximal_sequence_alldifferent`

Origin	N. Beldiceanu
Constraint	<code>size_maximal_sequence_alldifferent(SIZE, VARIABLES)</code>
Synonym(s)	<code>size_maximal_sequence_alldiff</code> , <code>size_maximal_sequence_alldistinct</code> .
Argument(s)	<code>SIZE</code> : <code>dvar</code> <code>VARIABLES</code> : <code>collection(var - dvar)</code>
Restriction(s)	<code>SIZE</code> \geq 0 <code>SIZE</code> \leq <code> VARIABLES </code> <code>required(VARIABLES, var)</code>
Purpose	<code>SIZE</code> is the size of the maximal sequence (among all sequences of consecutive variables of the collection <code>VARIABLES</code>) for which the <code>alldifferent</code> constraint holds.
Arc input(s)	<code>VARIABLES</code>
Arc generator	<code>PATH_N</code> \mapsto <code>collection</code>
Arc arity	*
Arc constraint(s)	<code>alldifferent(collection)</code>
Graph property(ies)	<code>NARC = SIZE</code>
Example	$\text{size_maximal_sequence_alldifferent} \left(4, \left\{ \begin{array}{l} \text{var} - 2, \\ \text{var} - 2, \\ \text{var} - 4, \\ \text{var} - 5, \\ \text{var} - 2, \\ \text{var} - 7, \\ \text{var} - 4 \end{array} \right\} \right)$ <p>The previous constraint holds since the constraint <code>alldifferent(var - 4, var - 5, var - 2, var - 7)</code> holds and since the following three constraints do not hold:</p> <ul style="list-style-type: none"> • <code>alldifferent(var - 2, var - 2, var - 4, var - 5, var - 2)</code>, • <code>alldifferent(var - 2, var - 4, var - 5, var - 2, var - 7)</code>, • <code>alldifferent(var - 4, var - 5, var - 2, var - 7, var - 4)</code>.
Graph model	Observe that this is an example of global constraint where the arc constraints don't have the same arity. However they correspond to the same type of constraint.
See also	<code>alldifferent</code> , <code>size_maximal_starting_sequence_alldifferent</code> .
Key words	sliding sequence constraint, conditional constraint, sequence, hypergraph.

20030820

783

4.184 size_maximal_starting_sequence_alldifferent

Origin	N. Beldiceanu
Constraint	size_maximal_starting_sequence_alldifferent(SIZE, VARIABLES)
Synonym(s)	size_maximal_starting_sequence_alldiff, size_maximal_starting_sequence_alldistinct.
Argument(s)	SIZE : dvar VARIABLES : collection(var – dvar)
Restriction(s)	SIZE ≥ 0 SIZE ≤ VARIABLES required(VARIABLES, var)
Purpose	SIZE is the size of the maximal sequence (among all sequences of consecutive variables of the collection VARIABLES starting at position one) for which the <code>alldifferent</code> constraint holds.
Arc input(s)	VARIABLES
Arc generator	<i>PATH_1</i> ↦ collection
Arc arity	*
Arc constraint(s)	alldifferent(collection)
Graph property(ies)	NARC = SIZE
Example	$\text{size_maximal_starting_sequence_alldifferent} \left(4, \left\{ \begin{array}{l} \text{var} - 9, \\ \text{var} - 2, \\ \text{var} - 4, \\ \text{var} - 5, \\ \text{var} - 2, \\ \text{var} - 7, \\ \text{var} - 4 \end{array} \right\} \right)$ <p>The previous constraint holds since the constraint <code>alldifferent(var – 9, var – 2, var – 4, var – 5)</code> holds and since <code>alldifferent(var – 9, var – 2, var – 4, var – 5, var – 2)</code> does not hold. Parts (A) and (B) of Figure 4.369 respectively show the initial and final graph.</p>
Graph model	Observe that this is an example where the arc constraints don't have the same arity. However they correspond to the same constraint.
Remark	A <i>conditional constraint</i> [145] with the specific structure that one can relax the constraints on the last variables of the collection VARIABLES.
See also	alldifferent, size_maximal_sequence_alldifferent.
Key words	sliding sequence constraint, conditional constraint, sequence, hypergraph.

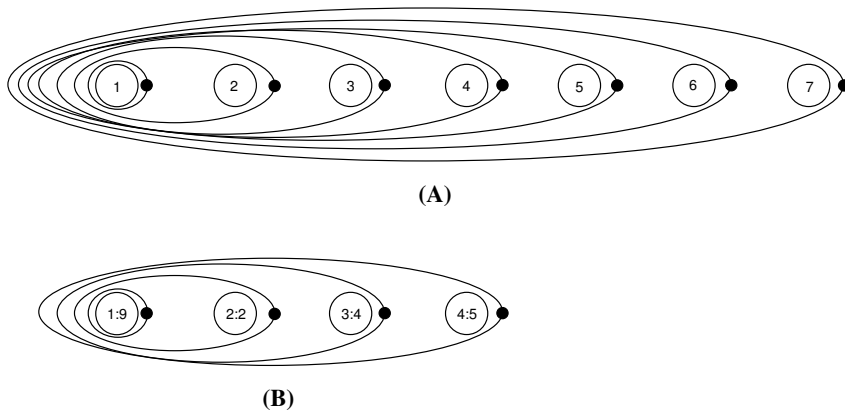


Figure 4.369: Initial and final graph of the `size_maximal_starting_sequence_alldifferent` constraint

4.185 sliding_card_skip0

Origin	N. Beldiceanu
Constraint	sliding_card_skip0(ATLEAST, ATMOST, VARIABLES, VALUES)
Argument(s)	ATLEAST : int ATMOST : int VARIABLES : collection(var – dvar) VALUES : collection(val – int)
Restriction(s)	ATLEAST ≥ 0 ATMOST \geq ATLEAST required(VARIABLES, var) required(VALUES, val) distinct(VALUES, val) VALUES.val $\neq 0$
Purpose	<div style="border: 1px solid black; padding: 5px;"> <p>Let n be the total number of variables of the collection VARIABLES. A <i>maximum non-zero set of consecutive variables</i> $X_i..X_j$ ($1 \leq i \leq j \leq n$) is defined in the following way:</p> <ul style="list-style-type: none"> • All variables X_i, \dots, X_j take a non-zero value, • $i = 1$ or X_{i-1} is equal to 0, • $j = n$ or X_{j+1} is equal to 0. <p>Enforces that each maximum non-zero set of consecutive variables of the collection VARIABLES contains at least ATLEAST and at most ATMOST values from the collection of values VALUES.</p> </div>
Arc input(s)	VARIABLES
Arc generator	PATH \mapsto collection(variables1, variables2) LOOP \mapsto collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	<ul style="list-style-type: none"> • variables1.var $\neq 0$ • variables2.var $\neq 0$
Sets	CC \mapsto [variables]
Constraint(s) on sets	among_low_up(ATLEAST, ATMOST, variables, VALUES)

Example	$\text{sliding_card_skip0} \left(2, 3, \left\{ \begin{array}{l} \text{var} - 0, \\ \text{var} - 7, \\ \text{var} - 2, \\ \text{var} - 9, \\ \text{var} - 0, \\ \text{var} - 0, \\ \text{var} - 9, \\ \text{var} - 4, \\ \text{var} - 9 \end{array} \right\}, \{ \text{val} - 7, \text{val} - 9 \} \right)$
----------------	---

Parts (A) and (B) of Figure 4.370 respectively show the initial and final graph. Since we use the set generator CC we show the two connected components of the final graph. Since these two connected components both contains between 2 and 3 variables which take there value in $\{7, 9\}$ the `sliding_card_skip0` constraint holds.

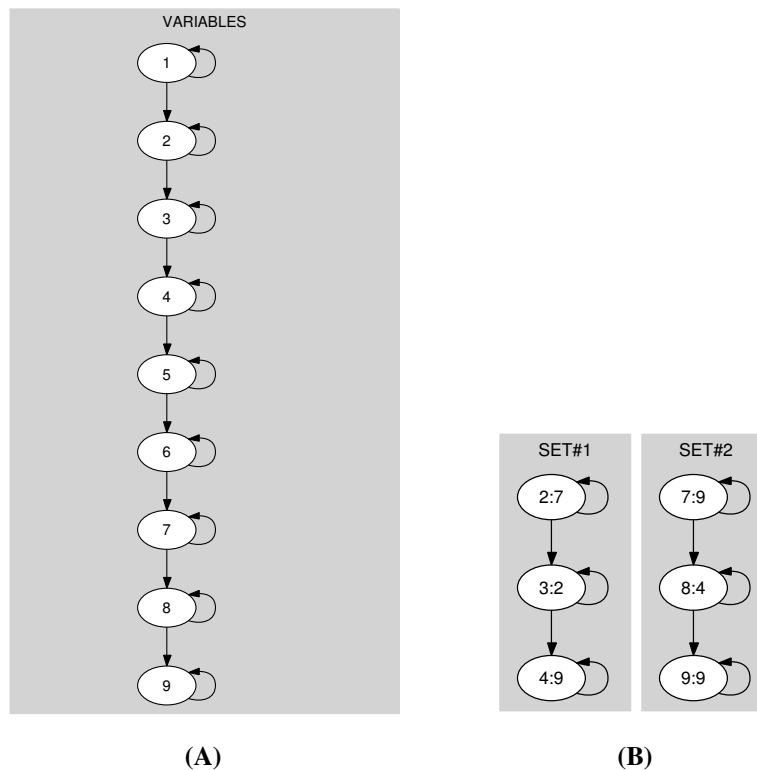


Figure 4.370: Initial and final graph of the `sliding_card_skip0` constraint

Graph model

Note that the arc constraint will produce the different sequences of consecutives variables that do not contain any 0. The CC set generator produces all the connected components of the final graph.

Automaton

Figure 4.371 depicts the automaton associated to the `sliding_card_skip0` constraint. To each variable VAR_i of the collection `VARIABLES` corresponds a signature variable S_i . The following signature constraint links VAR_i and S_i :

$$\begin{aligned} (VAR_i = 0) &\Leftrightarrow S_i = 0 \wedge \\ (VAR_i \neq 0 \wedge VAR_i \notin \text{VALUES}) &\Leftrightarrow S_i = 1 \wedge \\ (VAR_i \neq 0 \wedge VAR_i \in \text{VALUES}) &\Leftrightarrow S_i = 2. \end{aligned}$$

Usage

This constraint is useful in timetabling problems where the variables are interpreted as the type of job that a person does on consecutive days. Value 0 represents a rest day and one imposes a cardinality constraint on periods that are located between rest periods.

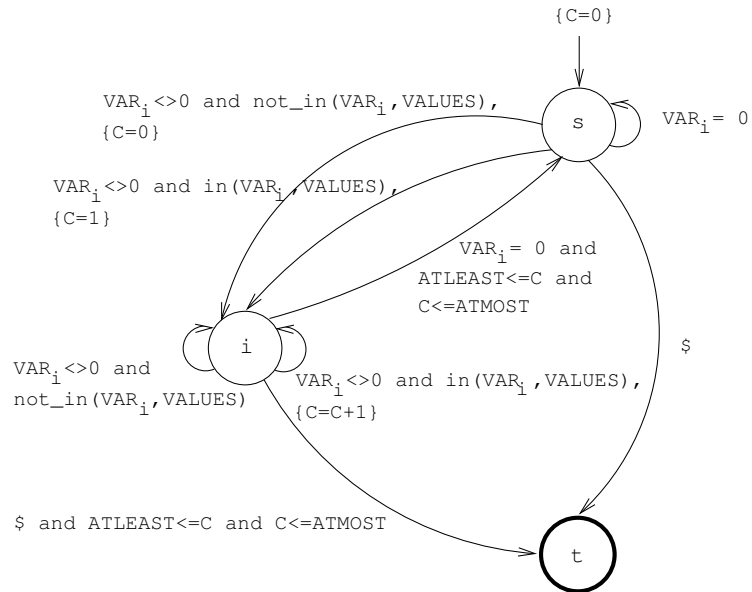


Figure 4.371: Automaton of the sliding_card_skip0 constraint

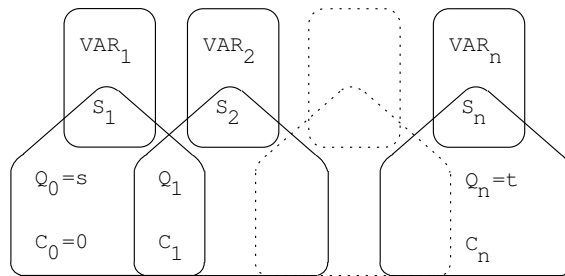


Figure 4.372: Hypergraph of the reformulation corresponding to the automaton of the sliding_card_skip0 constraint

- Remark** One cannot initially state a `global_cardinality` constraint since the rest days are not yet allocated. One can also not use an `among_seq` constraint since it does not hold for the sequences of consecutive variables that contains at least one rest day.
- See also** `among`, `among_low_up`, `global_cardinality`.
- Key words** timetabling constraint, sliding sequence constraint, sequence, automaton, automaton with counters, alpha-acyclic constraint network(2).

4.186 sliding_distribution

Origin	[146]
Constraint	<code>sliding_distribution(SEQ, VARIABLES, VALUES)</code>
Argument(s)	<pre> SEQ : int VARIABLES : collection(var - dvar) VALUES : collection(val - int, omin - int, omax - int) </pre>
Restriction(s)	<pre> SEQ > 0 SEQ ≤ VARIABLES required(VARIABLES, var) VALUES > 0 required(VALUES, [val, omin, omax]) distinct(VALUES, val) VALUES.omin ≥ 0 VALUES.omax ≤ SEQ VALUES.omin ≤ VALUES.omax </pre>
Purpose	<p>For each sequence of SEQ consecutive variables of the VARIABLES collection, each value $\text{VALUES}[i].\text{val}$ ($1 \leq i \leq \text{VALUES}$) should be taken by at least $\text{VALUES}[i].\text{omin}$ and at most $\text{VALUES}[i].\text{omax}$ variables.</p>
Arc input(s)	VARIABLES
Arc generator	$\text{PATH} \mapsto \text{collection}$
Arc arity	SEQ
Arc constraint(s)	<code>global_cardinality_low_up(collection, VALUES)</code>
Graph property(ies)	$\text{NARC} = \text{VARIABLES} - \text{SEQ} + 1$

Example	$\text{sliding_distribution} \left(4, \left\{ \begin{array}{l} \text{var} - 0, \\ \text{var} - 5, \\ \text{var} - 6, \\ \text{var} - 6, \\ \text{var} - 5, \\ \text{var} - 0, \\ \text{var} - 0 \end{array} \right\}, \left\{ \begin{array}{l} \text{val} - 0 \quad \text{omin} - 1 \quad \text{omax} - 2, \\ \text{val} - 1 \quad \text{omin} - 0 \quad \text{omax} - 4, \\ \text{val} - 4 \quad \text{omin} - 0 \quad \text{omax} - 4, \\ \text{val} - 5 \quad \text{omin} - 1 \quad \text{omax} - 2, \\ \text{val} - 6 \quad \text{omin} - 0 \quad \text{omax} - 2 \end{array} \right\} \right)$
----------------	---

The `sliding_distribution` constraint holds since:

- On the first sequence of 4 consecutive variables 0566 values 0, 1, 4, 5 and 6 are respectively used 1, 0, 0, 1 and 2 times.
- On the second sequence of 4 consecutive variables 5665 values 0, 1, 4, 5 and 6 are respectively used 0, 0, 0, 2 and 2 times.
- On the third sequence of 4 consecutive variables 6650 values 0, 1, 4, 5 and 6 are respectively used 1, 0, 0, 1 and 2 times.
- On the third sequence of 4 consecutive variables 6500 values 0, 1, 4, 5 and 6 are respectively used 2, 0, 0, 1 and 1 times.

See also `among_seq`, `global_cardinality_low_up`, `pattern`.

Key words decomposition, sliding sequence constraint, sequence, hypergraph.

4.187 sliding_sum

Origin	CHIP
Constraint	<code>sliding_sum(LOW, UP, SEQ, VARIABLES)</code>
Argument(s)	<pre> LOW : int UP : int SEQ : int VARIABLES : collection(var - dvar) </pre>
Restriction(s)	<pre> UP ≥ LOW SEQ > 0 SEQ ≤ VARIABLES required(VARIABLES, var) </pre>
Purpose	Constrains all sequences of SEQ consecutive variables of the collection VARIABLES so that the sum of the variables belongs to interval [LOW, UP].
Arc input(s)	VARIABLES
Arc generator	$\text{PATH} \mapsto \text{collection}$
Arc arity	SEQ
Arc constraint(s)	<ul style="list-style-type: none"> • <code>sum_ctr(collection, ≥, LOW)</code> • <code>sum_ctr(collection, ≤, UP)</code>
Graph property(ies)	$\overline{\text{NARC}} = \text{VARIABLES} - \text{SEQ} + 1$
Example	$\text{sliding_sum} \left(3, 7, 4, \left\{ \begin{array}{l} \text{var} - 1, \\ \text{var} - 4, \\ \text{var} - 2, \\ \text{var} - 0, \\ \text{var} - 0, \\ \text{var} - 3, \\ \text{var} - 4 \end{array} \right\} \right)$ <p>The previous example considers all sliding sequences of 4 consecutive variables and constraints the sum to be between 3 and 7. The constraint holds since the sum associated to the different sequences are respectively 7, 6, 5 and 7.</p>
Graph model	We use <code>sum_ctr</code> as an arc constraint. <code>sum_ctr</code> takes a collection of domain variables as its first argument.
Signature	Since we use the <i>PATH</i> arc generator with an arity of SEQ on the items of the VARIABLES collection, the expression $ \text{VARIABLES} - \text{SEQ} + 1$ corresponds to the maximum number of arcs of the final graph. Therefore we can rewrite the graph property $\overline{\text{NARC}} = \text{VARIABLES} - \text{SEQ} + 1$ to $\overline{\text{NARC}} \geq \text{VARIABLES} - \text{SEQ} + 1$ and simplify $\overline{\text{NARC}}$ to $\overline{\text{NARC}}$.

Algorithm [65].

Key words decomposition, sliding sequence constraint, sequence, hypergraph, sum.

4.188 sliding_time_window

Origin	N. Beldiceanu
Constraint	<code>sliding_time_window(WINDOW_SIZE, LIMIT, TASKS)</code>
Argument(s)	<code>WINDOW_SIZE : int</code> <code>LIMIT : int</code> <code>TASKS : collection(id - int, origin - dvar, duration - dvar)</code>
Restriction(s)	<code>WINDOW_SIZE > 0</code> <code>LIMIT ≥ 0</code> <code>required(TASKS, [id, origin, duration])</code> <code>distinct(TASKS, id)</code> <code>TASKS.duration ≥ 0</code>
Purpose	For any time window of size WINDOW_SIZE, the intersection of all the tasks of the collection TASKS with this time window is less than or equal to a given limit LIMIT.
Arc input(s)	TASKS
Arc generator	$CLIQUE \mapsto \text{collection}(\text{tasks1}, \text{tasks2})$
Arc arity	2
Arc constraint(s)	<ul style="list-style-type: none"> • $\text{tasks1.origin} \leq \text{tasks2.origin}$ • $\text{tasks2.origin} - \text{tasks1.origin} < \text{WINDOW_SIZE}$
Sets	$SUCC \mapsto [\text{source}, \text{tasks}]$
Constraint(s) on sets	<code>sliding_time_window_from_start(WINDOW_SIZE, LIMIT, tasks, source.origin)</code>

Example	$\text{sliding_time_window} \left(9, 6, \left\{ \begin{array}{l} \text{id} - 1 \quad \text{origin} - 10 \quad \text{duration} - 3, \\ \text{id} - 2 \quad \text{origin} - 5 \quad \text{duration} - 1, \\ \text{id} - 3 \quad \text{origin} - 6 \quad \text{duration} - 2, \\ \text{id} - 4 \quad \text{origin} - 14 \quad \text{duration} - 2, \\ \text{id} - 5 \quad \text{origin} - 2 \quad \text{duration} - 2 \end{array} \right\} \right)$
----------------	---

Parts (A) and (B) of Figure 4.373 respectively show the initial and final graph. In the final graph, the successors of a given task t correspond to the set of tasks that do not start before task t and intersect the time window that starts at the origin of task t .

The lower part of Figure 4.374 indicates the different tasks on the time axis. Each task is drawn as a rectangle with its corresponding identifier in the middle. Finally the upper part of Figure 4.374 shows the different time windows and the respective contribution of the tasks in these time windows. A line with two arrows depicts each time window. The two arrows indicate the start and the end of the time window. At the right of each time window we give its occupation. Since this occupation is always less than or equal to the limit 6, the `sliding_time_window` constraint holds.

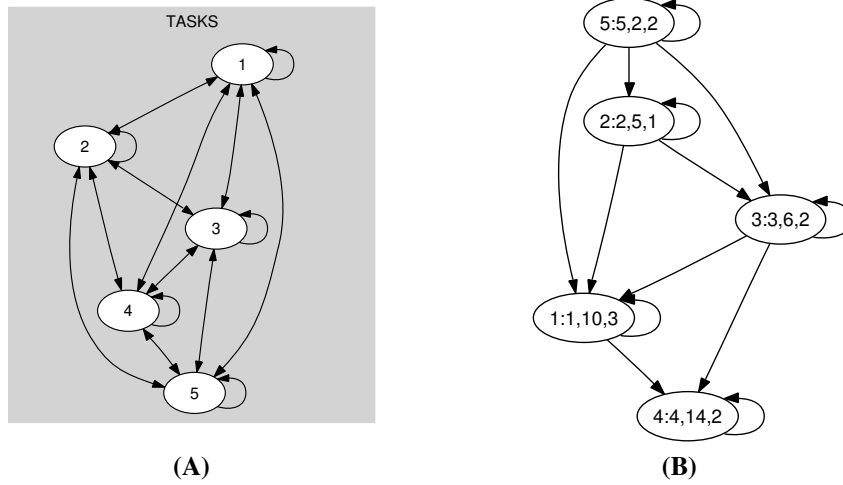


Figure 4.373: Initial and final graph of the sliding_time_window constraint

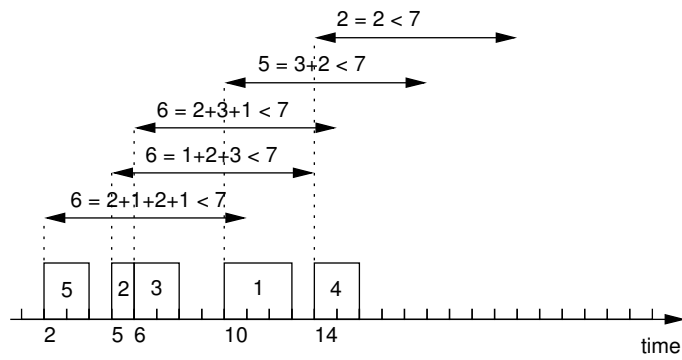


Figure 4.374: Time windows of the sliding_time_window constraint

Graph model	We generate an arc from a task t_1 to a task t_2 if task t_2 does not start before task t_1 and if task t_2 intersects the time window that starts at the origin of task t_1 . Each set generated by SUCC corresponds to all tasks that intersect in time the time window that starts at the origin of a given task.
Usage	The <code>sliding_time_window</code> constraint is useful for timetabling problems in order to put an upper limit on the total work over sliding time windows.
See also	<code>shift</code> , <code>sliding_time_window_from_start</code> , <code>sliding_time_window_sum</code> .
Key words	sliding sequence constraint, temporal constraint.

20030820

797

4.189 sliding_time_window_from_start

Origin	Used for defining <code>sliding_time_window</code> .
Constraint	<code>sliding_time_window_from_start(WINDOW_SIZE, LIMIT, TASKS, START)</code>
Argument(s)	<pre>WINDOW_SIZE : int LIMIT : int TASKS : collection(id - int, origin - dvar, duration - dvar) START : dvar</pre>
Restriction(s)	<pre>WINDOW_SIZE > 0 LIMIT ≥ 0 required(TASKS, [id, origin, duration]) distinct(TASKS, id) TASKS.duration ≥ 0</pre>
Purpose	The sum of the intersections of all the tasks of the <code>TASKS</code> collection with interval <code>[START, START + WINDOW_SIZE - 1]</code> is less than or equal to <code>LIMIT</code> .
Derived Collection(s)	<code>col(S - collection(var - dvar), [item(var - START)])</code>
Arc input(s)	<code>S TASKS</code>
Arc generator	<code>PRODUCT</code> \mapsto <code>collection(s, tasks)</code>
Arc arity	2
Arc constraint(s)	TRUE
Graph property(ies)	$\text{SUM_WEIGHT_ARC} \left(\max \left(0, \frac{\min(s.\text{var} + \text{WINDOW_SIZE}, \text{tasks}.\text{origin} + \text{tasks}.\text{duration}) - \max(s.\text{var}, \text{tasks}.\text{origin})}{} \right) \right)$
Example	$\text{sliding_time_window} \left(9, 6, \left\{ \begin{array}{lll} \text{id} - 1 & \text{origin} - 10 & \text{duration} - 3, \\ \text{id} - 2 & \text{origin} - 5 & \text{duration} - 1, \\ \text{id} - 3 & \text{origin} - 6 & \text{duration} - 2 \end{array} \right\}, 5 \right)$ <p>Parts (A) and (B) of Figure 4.375 respectively show the initial and final graph. To each arc of the final graph we associate the intersection of the corresponding sink task with interval <code>[START, START + WINDOW_SIZE - 1]</code>. The constraint <code>sliding_time_window_from_start</code> holds since the sum of the previous intersections does not exceed <code>LIMIT</code>.</p>
Graph model	Since we use the <code>TRUE</code> arc constraint the final and the initial graph are identical. The unique source of the final graph corresponds to the interval <code>[START, START + WINDOW_SIZE - 1]</code> . Each sink of the final graph represents a given task of the <code>TASKS</code> collection. We value each arc by the intersection of the task associated to one of the extremities of the arc with the time window <code>[START, START + WINDOW_SIZE - 1]</code> . Finally, the graph property <code>SUM_WEIGHT_ARC</code> sums up all the valuations of the arcs and check that it does not exceed a given limit.

20030820

799

Used in `sliding_time_window`.

Key words sliding sequence constraint, temporal constraint, derived collection.

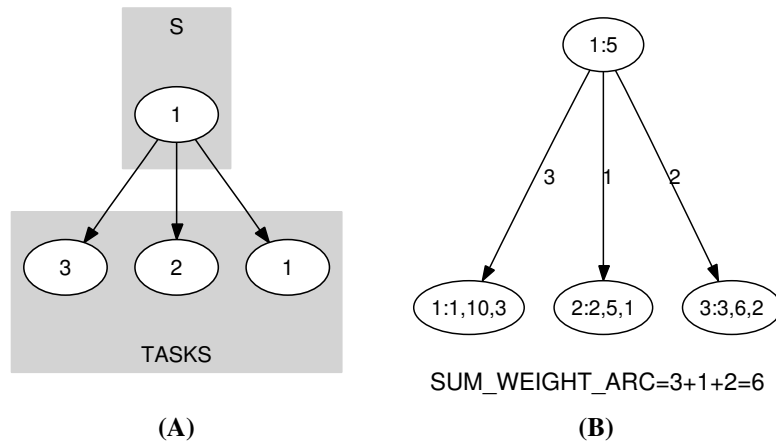


Figure 4.375: Initial and final graph of the `sliding_time_window_from_start` constraint

20030820

801

4.190 sliding_time_window_sum

Origin	Derived from <code>sliding_time_window</code> .
Constraint	<code>sliding_time_window_sum(WINDOW_SIZE, LIMIT, TASKS)</code>
Argument(s)	<p><code>WINDOW_SIZE</code> : int</p> <p><code>LIMIT</code> : int</p> <p><code>TASKS</code> : <code>collection(id - int, origin - dvar, end - dvar, npoint - dvar)</code></p>
Restriction(s)	<p><code>WINDOW_SIZE > 0</code></p> <p><code>LIMIT ≥ 0</code></p> <p><code>required(TASKS, [id, origin, end, npoint])</code></p> <p><code>distinct(TASKS, id)</code></p> <p><code>TASKS.npoint ≥ 0</code></p>

Purpose

For any time window of size `WINDOW_SIZE`, the sum of the points of the tasks of the collection `TASKS` that overlap that time window do not exceed a given limit `LIMIT`.

Arc input(s)	<code>TASKS</code>
Arc generator	<code>SELF</code> \mapsto <code>collection(tasks)</code>
Arc arity	1
Arc constraint(s)	<code>tasks.origin ≤ tasks.end</code>
Graph property(ies)	$\overline{\text{NARC}} = \text{TASKS} $

Arc input(s)	<code>TASKS</code>
Arc generator	<code>CLIQUE</code> \mapsto <code>collection(tasks1, tasks2)</code>
Arc arity	2
Arc constraint(s)	<ul style="list-style-type: none"> • <code>tasks1.end ≤ tasks2.end</code> • <code>tasks2.origin - tasks1.end < WINDOW_SIZE - 1</code>
Sets	<p><code>SUCC</code> \mapsto</p> $\left[\begin{array}{l} \text{source,} \\ \text{variables - col} \left(\begin{array}{l} \text{VARIABLES - collection(var - dvar),} \\ \text{[item(var - TASKS.npoint)]} \end{array} \right) \end{array} \right]$
Constraint(s) on sets	<code>sum_ctr(variables, ≤, LIMIT)</code>

Example

$$\text{sliding_time_window_sum} \left(9, 16, \left\{ \begin{array}{l} \text{id} - 1 \quad \text{origin} - 10 \quad \text{end} - 13 \quad \text{npoint} - 2, \\ \text{id} - 2 \quad \text{origin} - 5 \quad \text{end} - 6 \quad \text{npoint} - 3, \\ \text{id} - 3 \quad \text{origin} - 6 \quad \text{end} - 8 \quad \text{npoint} - 4, \\ \text{id} - 4 \quad \text{origin} - 14 \quad \text{end} - 16 \quad \text{npoint} - 5, \\ \text{id} - 5 \quad \text{origin} - 2 \quad \text{end} - 4 \quad \text{npoint} - 6 \end{array} \right\} \right)$$

Parts (A) and (B) of Figure 4.376 respectively show the initial and final graph. In the final graph, the successors of a given task t correspond to the set of tasks that both do not end before the end of task t , and intersect the time window that starts at the end - 1 of task t .

The lower part of Figure 4.377 indicates the different tasks on the time axis. Each task is drawn as a rectangle with its corresponding identifier in the middle. Finally the upper part of Figure 4.377 shows the different time windows and the respective contribution of the tasks in these time windows. A line with two arrows depicts each time window. The two arrows indicate the start and the end of the time window. At the right of each time window we give its occupation. Since this occupation is always less than or equal to the limit 16, the `sliding_time_window_sum` constraint holds.

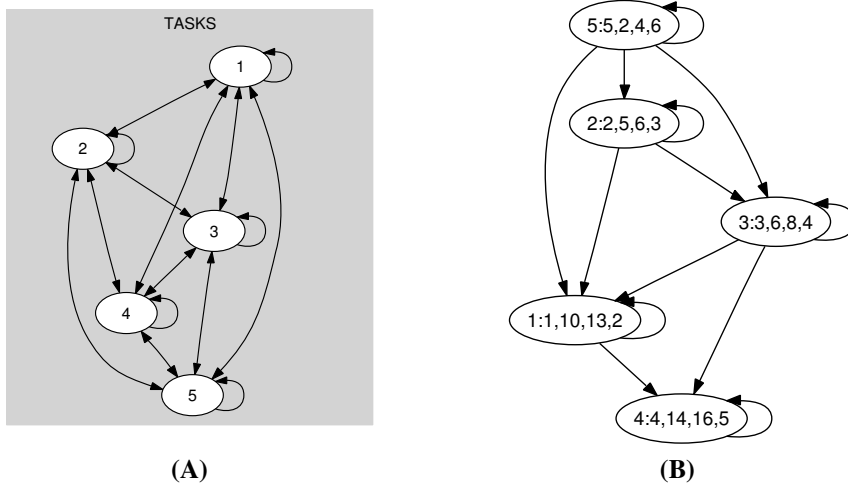


Figure 4.376: Initial and final graph of the `sliding_time_window_sum` constraint

Graph model

We generate an arc from a task t_1 to a task t_2 if task t_2 does not end before the end of task t_1 and if task t_2 intersects the time window that starts at the last instant of task t_1 . Each set generated by `SUCC` corresponds to all tasks that intersect in time the time window that starts at instant `end - 1`, where `end` is the end of a given task.

Signature

Consider the first graph constraint. Since we use the `SELF` arc generator on the `TASKS` collection the maximum number of arcs of the final graph is equal to $|\text{TASKS}|$. Therefore we can rewrite $\text{NARC} = |\text{TASKS}|$ to $\text{NARC} \geq |\text{TASKS}|$ and simplify NARC to $\overline{\text{NARC}}$.

Usage

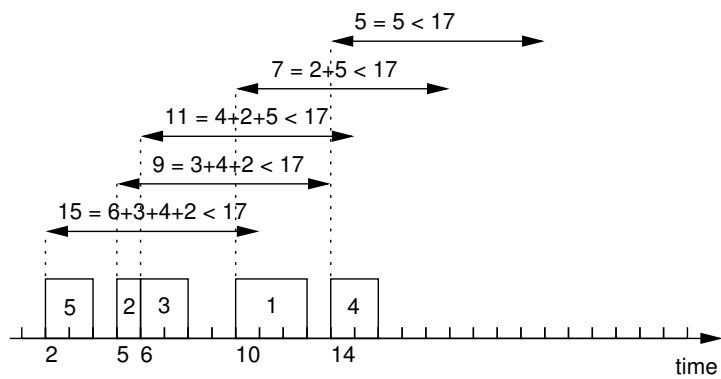
This constraint may be used for timetabling problems in order to put an upper limit on the cumulated number of points in a shift.

See also

`sliding_time_window`.

Key words

sliding sequence constraint, temporal constraint, time window, sum.

Figure 4.377: Time windows of the `sliding_time_window_sum` constraint

20030820

805

4.191 smooth

Origin	Derived from change.
Constraint	<code>smooth(NCHANGE, TOLERANCE, VARIABLES)</code>
Argument(s)	<code>NCHANGE : dvar</code> <code>TOLERANCE : int</code> <code>VARIABLES : collection(var - dvar)</code>
Restriction(s)	<code>NCHANGE ≥ 0</code> <code>NCHANGE < VARIABLES </code> <code>TOLERANCE ≥ 0</code> <code>required(VARIABLES, var)</code>
Purpose	NCHANGE is the number of times that $ X - Y > \text{TOLERANCE}$ holds; X and Y correspond to consecutive variables of the collection VARIABLES.
Arc input(s)	VARIABLES
Arc generator	$PATH \mapsto \text{collection}(\text{variables1}, \text{variables2})$
Arc arity	2
Arc constraint(s)	<code>abs(variables1.var - variables2.var) > TOLERANCE</code>
Graph property(ies)	<code>NARC = NCHANGE</code>
Example	$\text{smooth} \left(1, 2, \left\{ \begin{array}{l} \text{var} - 1, \\ \text{var} - 3, \\ \text{var} - 4, \\ \text{var} - 5, \\ \text{var} - 2 \end{array} \right\} \right)$ <p>In the previous example we have one change between values 5 and 2 since the difference in absolute value is greater than the tolerance (i.e. $5 - 2 > 2$). Parts (A) and (B) of Figure 4.378 respectively show the initial and final graph. Since we use the NARC graph property, the unique arc of the final graph is stressed in bold.</p>
Automaton	Figure 4.379 depicts the automaton associated to the <code>smooth</code> constraint. To each pair of consecutive variables $(\text{VAR}_i, \text{VAR}_{i+1})$ of the collection VARIABLES corresponds a 0-1 signature variable S_i . The following signature constraint links VAR_i , VAR_{i+1} and S_i : $(\text{VAR}_i - \text{VAR}_{i+1} > \text{TOLERANCE} \Leftrightarrow S_i = 1$.
Usage	This constraint is useful for the following problems: <ul style="list-style-type: none"> • Assume that VARIABLES corresponds to the number of people that work on consecutive weeks. One may not normally increase or decrease too drastically the number of people from one week to the next week. With the <code>smooth</code> constraint you can state a limit on the number of drastic changes.

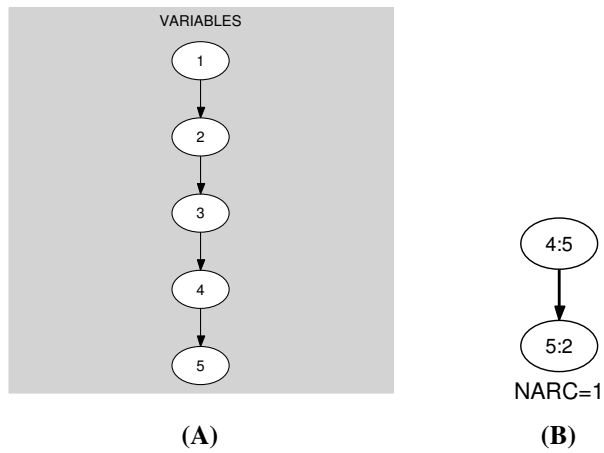


Figure 4.378: Initial and final graph of the smooth constraint

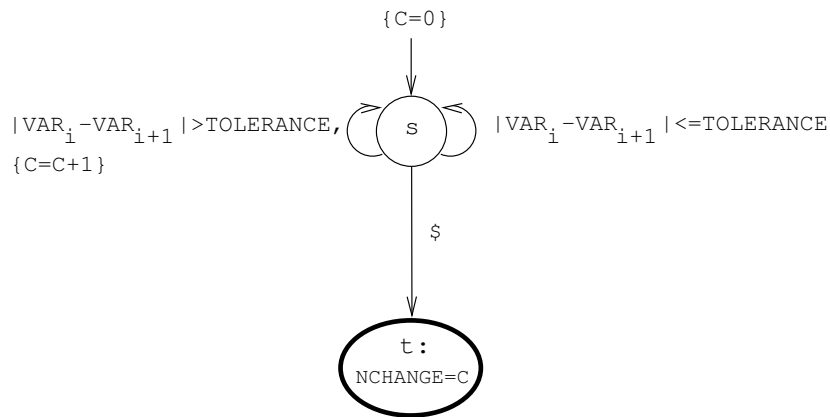


Figure 4.379: Automaton of the smooth constraint

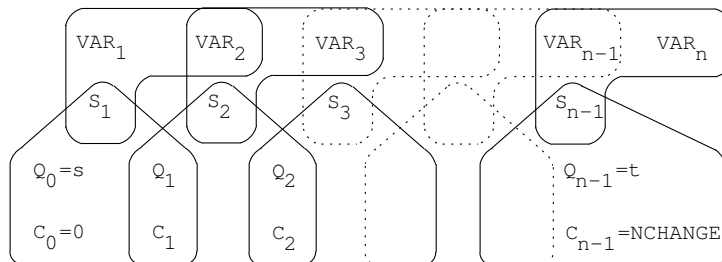


Figure 4.380: Hypergraph of the reformulation corresponding to the automaton of the smooth constraint

- Assume you have to produce a set of orders, each order having a specific attribute. You want to generate the orders in such a way that there is not a too big difference between the values of the attributes of two consecutive orders. If you can't achieve this on two given specific orders, this would imply a set-up or a cost. Again, with the `smooth` constraint, you can control this kind of drastic changes.

Algorithm

[65].

See also`change`.**Key words**

timetabling constraint, number of changes, automaton, automaton with counters, sliding cyclic(1) constraint network(2).

20000128

809

4.192 soft_alldifferent_ctr

Origin	[10]
Constraint	<code>soft_alldifferent_ctr(C, VARIABLES)</code>
Synonym(s)	<code>soft_alldiff_ctr</code> , <code>soft_alldistinct_ctr</code> .
Argument(s)	<code>C</code> : dvar <code>VARIABLES</code> : collection(var – dvar)
Restriction(s)	$C \geq 0$ $C \leq (VARIABLES * VARIABLES - VARIABLES)/2$ <code>required(VARIABLES, var)</code>
Purpose	Consider the <i>disequality</i> constraints involving two distinct variables of the collection <code>VARIABLES</code> . Among the previous set of constraints, <code>C</code> is the number of <i>disequality</i> constraints which do not hold.
Arc input(s)	<code>VARIABLES</code>
Arc generator	<code>CLIQUE(<) ↦ collection(variables1, variables2)</code>
Arc arity	2
Arc constraint(s)	<code>variables1.var = variables2.var</code>
Graph property(ies)	<code>NARC = C</code>

Example `soft_alldifferent_ctr` $\left(4, \left\{ \begin{array}{l} \text{var} - 5, \\ \text{var} - 1, \\ \text{var} - 9, \\ \text{var} - 1, \\ \text{var} - 5, \\ \text{var} - 5 \end{array} \right\} \right)$

Parts (A) and (B) of Figure 4.381 respectively show the initial and final graph. Since we use the **NARC** graph property, the arcs of the final graph are stressed in bold. Since four equality constraints remain in the final graph the *cost* variable `C` is equal to 4.

Graph model We generate an initial graph with binary *equalities* constraints between each vertex and its successors. We use the arc generator `CLIQUE(<)` in order to avoid counting twice the same *equality* constraint. The graph property states that `C` is equal to the number of *equalities* that hold in the final graph.

Usage A soft `alldifferent` constraint.

- Algorithm** Since it focus on the soft aspect of the `alldifferent` constraint, the original paper [10] which introduces this constraint describes how to evaluate the minimum value of C and how to prune according to the maximum value of C . The corresponding filtering algorithm does not achieve arc-consistency. W.-J. van Hoeve [26] presents a new filtering algorithm which achieves arc-consistency. This algorithm is based on a reformulation into a minimum-cost flow problem.
- See also** `alldifferent`, `soft_alldifferent_var`.
- Key words** soft constraint, value constraint, relaxation, decomposition-based violation measure, all different, disequality, flow.

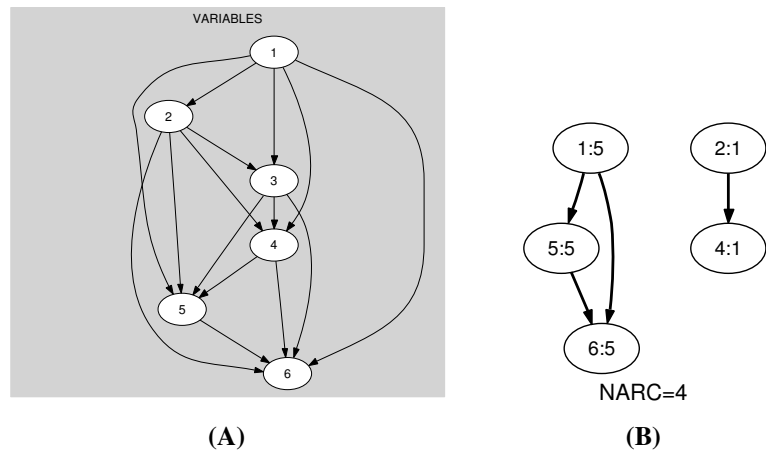
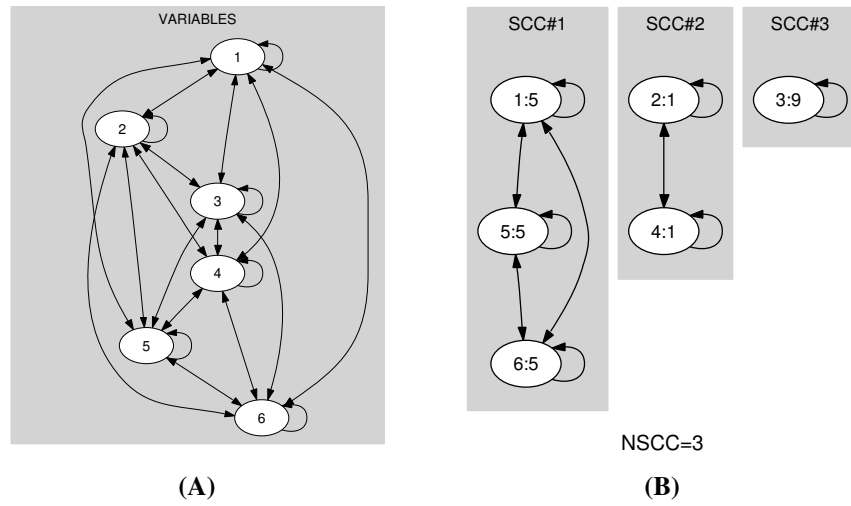


Figure 4.381: Initial and final graph of the soft_alldifferent_ctr constraint

4.193 soft_alldifferent_var

Origin	[10]
Constraint	<code>soft_alldifferent_var(C, VARIABLES)</code>
Synonym(s)	<code>soft_alldiff_var</code> , <code>soft_alldistinct_var</code> .
Argument(s)	<code>C</code> : dvar <code>VARIABLES</code> : collection(var – dvar)
Restriction(s)	$C \geq 0$ $C < \text{VARIABLES} $ <code>required(VARIABLES, var)</code>
Purpose	C is the minimum number of variables of the collection VARIABLES for which the value needs to be changed in order that all variables of VARIABLES take a distinct value.
<hr/>	
Arc input(s)	VARIABLES
Arc generator	<i>CLIQUE</i> \mapsto <code>collection(variables1, variables2)</code>
Arc arity	2
Arc constraint(s)	<code>variables1.var = variables2.var</code>
Graph property(ies)	$\text{NSCC} = \text{VARIABLES} - C$
<hr/>	
Example	$\text{soft_alldifferent_var} \left(3, \left\{ \begin{array}{l} \text{var} - 5, \\ \text{var} - 1, \\ \text{var} - 9, \\ \text{var} - 1, \\ \text{var} - 5, \\ \text{var} - 5 \end{array} \right\} \right)$ <p>Parts (A) and (B) of Figure 4.382 respectively show the initial and final graph. Since we use the NSCC graph property we show the different strongly connected components of the final graph. Each strongly connected component of the final graph includes all variables which take the same value. Since we have 6 variables and 3 strongly connected components the <i>cost</i> variable C is equal to 6 – 3.</p>
Graph model	We generate a clique with binary <i>equalities</i> constraints between each pairs of vertices (this include an arc between a vertex and itself) and we state that C is equal to the difference between the total number of variables and the number of strongly connected components.
Usage	A soft <code>alldifferent</code> constraint.
Remark	Since it focus on the soft aspect of the <code>alldifferent</code> constraint, the original paper [10] which introduce this constraint describes how to evaluate the minimum value of C and how to prune according to the maximum value of C.

- Algorithm** The filtering algorithm presented in [10] achieves arc-consistency.
- See also** `alldifferent`, `soft_alldifferent_ctr`, `weighted_partial_alldiff`.
- Key words** soft constraint, value constraint, relaxation, variable-based violation measure, all different, disequality, strongly connected component, equivalence.

Figure 4.382: Initial and final graph of the `soft_alldifferent_var` constraint

20030820

817

4.194 `soft_same_interval_var`

Origin	Derived from <code>same_interval</code>
Constraint	<code>soft_same_interval_var(C, VARIABLES1, VARIABLES2, SIZE_INTERVAL)</code>
Synonym(s)	<code>soft_same_interval.</code>
Argument(s)	<code>C</code> : <code>dvar</code> <code>VARIABLES1</code> : <code>collection(var - dvar)</code> <code>VARIABLES2</code> : <code>collection(var - dvar)</code> <code>SIZE_INTERVAL</code> : <code>int</code>
Restriction(s)	$C \geq 0$ $C \leq \text{VARIABLES1} $ $ \text{VARIABLES1} = \text{VARIABLES2} $ <code>required(VARIABLES1, var)</code> <code>required(VARIABLES2, var)</code> $\text{SIZE_INTERVAL} > 0$
Purpose	Let N_i (respectively M_i) denote the number of variables of the collection <code>VARIABLES1</code> (respectively <code>VARIABLES2</code>) that take a value in the interval $[\text{SIZE_INTERVAL} \cdot i, \text{SIZE_INTERVAL} \cdot i + \text{SIZE_INTERVAL} - 1]$. <code>C</code> is the minimum number of values to change in the <code>VARIABLES1</code> and <code>VARIABLES2</code> collections so that for all integer i we have $N_i = M_i$.
Arc input(s)	<code>VARIABLES1</code> <code>VARIABLES2</code>
Arc generator	<code>PRODUCT</code> \mapsto <code>collection(variables1, variables2)</code>
Arc arity	2
Arc constraint(s)	<code>variables1.var/SIZE_INTERVAL = variables2.var/SIZE_INTERVAL</code>
Graph property(ies)	<code>NSINK_NSOURCE = VARIABLES1 - C</code>

Example	<code>soft_same_interval_var</code> $\left(\begin{array}{c} \left\{ \begin{array}{c} \text{var} - 9, \\ \text{var} - 9, \\ \text{var} - 9, \\ \text{var} - 9, \\ \text{var} - 9, \\ \text{var} - 1 \end{array} \right\}, 4, \\ \left\{ \begin{array}{c} \text{var} - 9, \\ \text{var} - 1, \\ \text{var} - 1, \\ \text{var} - 1, \\ \text{var} - 1, \\ \text{var} - 8 \end{array} \right\}, 3 \end{array} \right)$
----------------	---

Parts (A) and (B) of Figure 4.383 respectively show the initial and final graph.

Since we use the `NSINK_NSOURCE` graph property, the source and sink vertices of the final graph are stressed with a double circle. The `soft_same_interval_var` constraint holds since the cost 4 corresponds to the difference between the number of variables of `VARIABLES1` and the sum over the different connected components of the minimum number of sources and sinks.

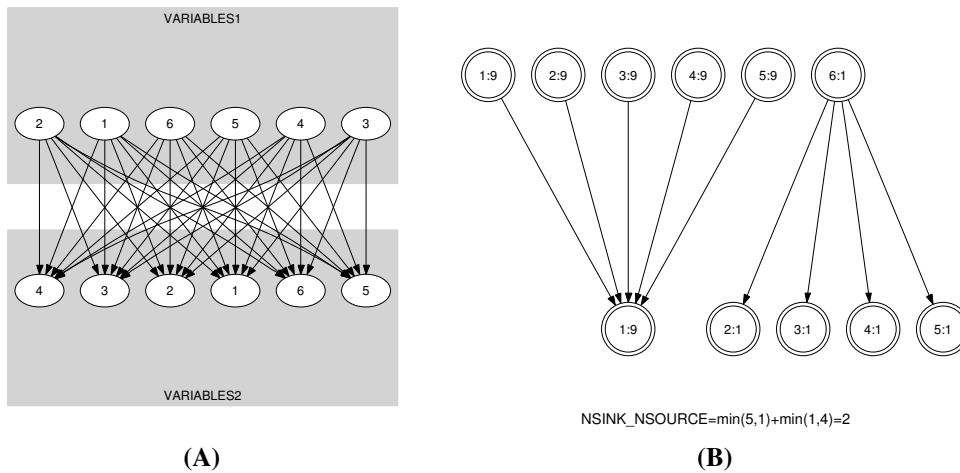


Figure 4.383: Initial and final graph of the `soft_same_interval_var` constraint

Usage	A soft <code>same_interval</code> constraint.
Algorithm	See algorithm of the <code>soft_same_var</code> constraint.
See also	<code>same_interval</code> .
Key words	soft constraint, constraint between two collections of variables, relaxation, variable-based violation measure, interval.

4.195 soft_same_modulo_var

Origin	Derived from same_modulo
Constraint	soft_same_modulo_var(C, VARIABLES1, VARIABLES2, M)
Synonym(s)	soft_same_modulo.
Argument(s)	C : dvar VARIABLES1 : collection(var - dvar) VARIABLES2 : collection(var - dvar) M : int

Restriction(s)	$C \geq 0$ $C \leq \text{VARIABLES1} $ $ \text{VARIABLES1} = \text{VARIABLES2} $ required(VARIABLES1, var) required(VARIABLES2, var) $M > 0$
-----------------------	--

Purpose

For each integer R in $[0, M - 1]$, let $N1_R$ (respectively $N2_R$) denote the number of variables of VARIABLES1 (respectively VARIABLES2) which have R as a rest when divided by M . C is the minimum number of values to change in the VARIABLES1 and VARIABLES2 collections so that for all R in $[0, M - 1]$ we have $N1_R = N2_R$.

Arc input(s)	VARIABLES1 VARIABLES2
Arc generator	<i>PRODUCT</i> \mapsto collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	variables1.var mod M = variables2.var mod M
Graph property(ies)	<u>NSINK_NSOURCE = VARIABLES1 - C</u>

Example

$$\text{soft_same_modulo_var} \left(4, \left\{ \begin{array}{l} \text{var} - 9, \\ \text{var} - 9, \\ \text{var} - 9, \\ \text{var} - 9, \\ \text{var} - 1 \end{array} \right\}, \left\{ \begin{array}{l} \text{var} - 9, \\ \text{var} - 1, \\ \text{var} - 1, \\ \text{var} - 1, \\ \text{var} - 1, \\ \text{var} - 8 \end{array} \right\}, 3 \right)$$

Parts (A) and (B) of Figure 4.384 respectively show the initial and final graph.

Since we use the `NSINK_NSOURCE` graph property, the source and sink vertices of the final graph are stressed with a double circle. The `soft_same_modulo_var` constraint holds since the cost 4 corresponds to the difference between the number of variables of `VARIABLES1` and the sum over the different connected components of the minimum number of sources and sinks.

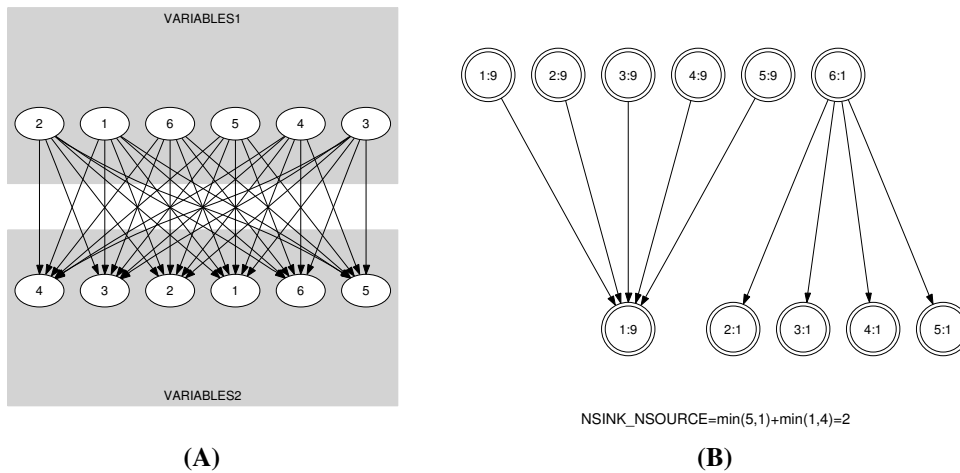


Figure 4.384: Initial and final graph of the `soft_same_modulo_var` constraint

Usage

A soft `same_modulo` constraint.

Algorithm

See algorithm of the `soft_same_var` constraint.

See also

`same_modulo`.

Key words

soft constraint, constraint between two collections of variables, relaxation, variable-based violation measure, modulo.

4.196 soft_same_partition_var

Origin	Derived from <code>same_partition</code>
Constraint	<code>soft_same_partition_var(C, VARIABLES1, VARIABLES2, PARTITIONS)</code>
Synonym(s)	<code>soft_same_partition</code> .
Type(s)	<code>VALUES : collection(val – int)</code>
Argument(s)	<code>C : dvar</code> <code>VARIABLES1 : collection(var – dvar)</code> <code>VARIABLES2 : collection(var – dvar)</code> <code>PARTITIONS : collection(p – VALUES)</code>
Restriction(s)	$C \geq 0$ $C \leq \text{VARIABLES1} $ $ \text{VARIABLES1} = \text{VARIABLES2} $ <code>required(VARIABLES1, var)</code> <code>required(VARIABLES2, var)</code> <code>required(PARTITIONS, p)</code> $ \text{PARTITIONS} \geq 2$ <code>required(VALUES, val)</code> <code>distinct(VALUES, val)</code>
Purpose	<p>For each integer i in $[1, \text{PARTITIONS}]$, let $N1_i$ (respectively $N2_i$) denote the number of variables of <code>VARIABLES1</code> (respectively <code>VARIABLES2</code>) which take their value in the i^{th} partition of the collection <code>PARTITIONS</code>. C is the minimum number of values to change in the <code>VARIABLES1</code> and <code>VARIABLES2</code> collections so that for all i in $[1, \text{PARTITIONS}]$ we have $N1_i = N2_i$.</p>
Arc input(s)	<code>VARIABLES1 VARIABLES2</code>
Arc generator	<code>PRODUCT</code> \mapsto <code>collection(variables1, variables2)</code>
Arc arity	2
Arc constraint(s)	<code>in_same_partition(variables1.var, variables2.var, PARTITIONS)</code>
Graph property(ies)	<u><code>NSINK_NSOURCE = \text{VARIABLES1} – C</code></u>

Example

$$\text{soft_same_partition_var} \left(4, \left\{ \begin{array}{l} \text{var} - 9, \\ \text{var} - 9, \\ \text{var} - 9, \\ \text{var} - 9, \\ \text{var} - 9, \\ \text{var} - 1 \end{array} \right\}, \left\{ \begin{array}{l} \text{var} - 9, \\ \text{var} - 1, \\ \text{var} - 1, \\ \text{var} - 1, \\ \text{var} - 1, \\ \text{var} - 8 \end{array} \right\}, \left\{ \begin{array}{l} p - \{\text{val} - 1, \text{val} - 2\}, \\ p - \{\text{val} - 9\}, \\ p - \{\text{val} - 7, \text{val} - 8\} \end{array} \right\} \right)$$

Parts (A) and (B) of Figure 4.385 respectively show the initial and final graph. Since we use the `NSINK_NSOURCE` graph property, the source and sink vertices of the final graph are stressed with a double circle. The `soft_same_partition_var` constraint holds since the cost 4 corresponds to the difference between the number of variables of `VARIABLES1` and the sum over the different connected components of the minimum number of sources and sinks.

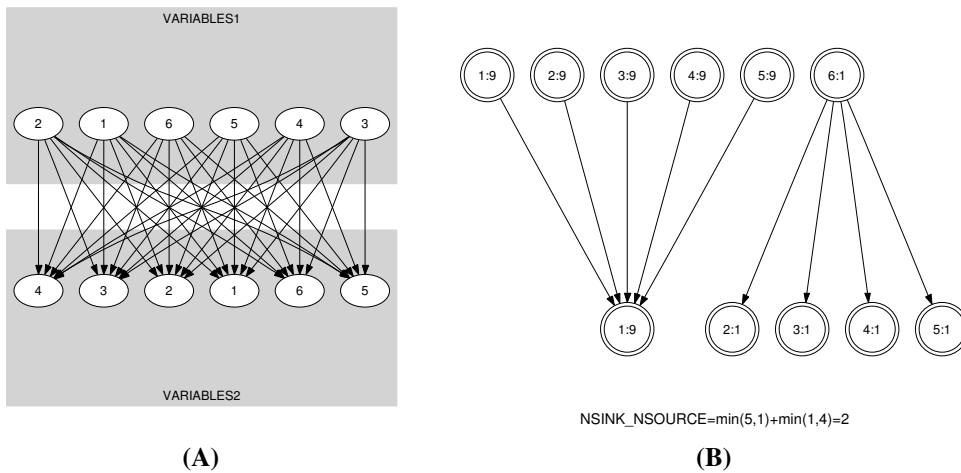


Figure 4.385: Initial and final graph of the `soft_same_partition_var` constraint

- Usage** A soft `same_partition` constraint.
- Algorithm** See algorithm of the `soft_same_var` constraint.
- See also** `same_partition`.
- Key words** soft constraint, constraint between two collections of variables, relaxation, variable-based violation measure, partition.

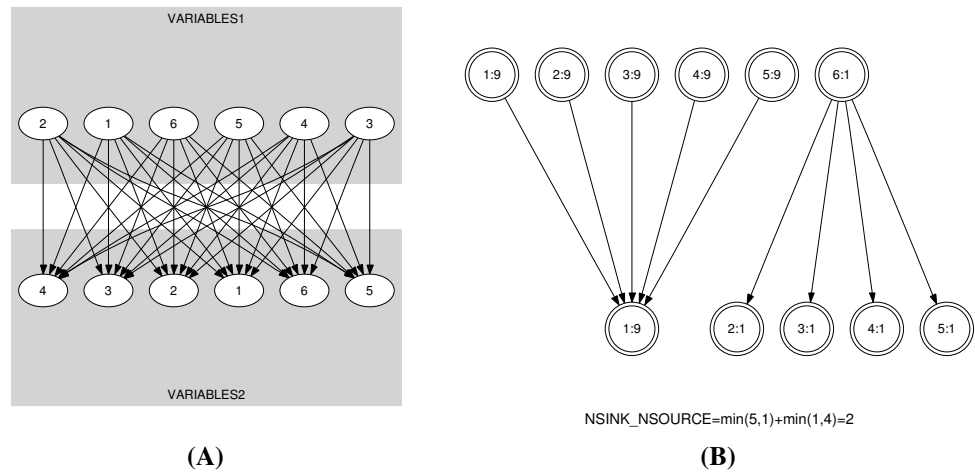
4.197 soft_same_var

Origin	[104]
Constraint	<code>soft_same_var(C, VARIABLES1, VARIABLES2)</code>
Synonym(s)	<code>soft_same.</code>
Argument(s)	<p><code>C</code> : <code>dvar</code> <code>VARIABLES1</code> : <code>collection(var - dvar)</code> <code>VARIABLES2</code> : <code>collection(var - dvar)</code></p>
Restriction(s)	<p>$C \geq 0$ $C \leq \text{VARIABLES1}$ $\text{VARIABLES1} = \text{VARIABLES2}$ <code>required(VARIABLES1, var)</code> <code>required(VARIABLES2, var)</code></p>
Purpose	<p><code>C</code> is the minimum number of values to change in the <code>VARIABLES1</code> and <code>VARIABLES2</code> collections so that the variables of the <code>VARIABLES2</code> collection correspond to the variables of the <code>VARIABLES1</code> collection according to a permutation.</p>
Arc input(s)	<code>VARIABLES1 VARIABLES2</code>
Arc generator	<code>PRODUCT</code> \mapsto <code>collection(variables1, variables2)</code>
Arc arity	2
Arc constraint(s)	<code>variables1.var = variables2.var</code>
Graph property(ies)	<code>NSINK_NSOURCE = VARIABLES1 - C</code>

Example	$\text{soft_same_var} \left(4, \left(\left(\begin{array}{c} \text{var} - 9, \\ \text{var} - 9, \\ \text{var} - 9, \\ \text{var} - 9, \\ \text{var} - 1 \end{array} \right), \left(\begin{array}{c} \text{var} - 9, \\ \text{var} - 1, \\ \text{var} - 1, \\ \text{var} - 1, \\ \text{var} - 1, \\ \text{var} - 8 \end{array} \right) \right) \right)$
----------------	---

Parts (A) and (B) of Figure 4.386 respectively show the initial and final graph. Since we use the `NSINK_NSOURCE` graph property, the source and sink vertices of the final graph are stressed with a double circle. The `soft_same_var` constraint holds since the cost 4 corresponds to the difference between the number of variables of `VARIABLES1` and the sum over the different connected components of the minimum number of sources and sinks.

Usage	A soft same constraint.
Algorithm	[104, page 80].
See also	same.
Key words	soft constraint, constraint between two collections of variables, relaxation, variable-based violation measure.

Figure 4.386: Initial and final graph of the `soft_same_var` constraint

20050507

827

4.198 `soft_used_by_interval_var`

Origin	Derived from <code>used_by_interval</code> .
Constraint	<code>soft_used_by_interval_var(C, VARIABLES1, VARIABLES2, SIZE_INTERVAL)</code>
Synonym(s)	<code>soft_used_by_interval</code> .
Argument(s)	<code>C</code> : dvar <code>VARIABLES1</code> : collection(var - dvar) <code>VARIABLES2</code> : collection(var - dvar) <code>SIZE_INTERVAL</code> : int
Restriction(s)	$C \geq 0$ $C \leq \text{VARIABLES2} $ $ \text{VARIABLES1} \geq \text{VARIABLES2} $ required(VARIABLES1, var) required(VARIABLES2, var) $\text{SIZE_INTERVAL} > 0$
Purpose	Let N_i (respectively M_i) denote the number of variables of the collection VARIABLES1 (respectively VARIABLES2) that take a value in the interval $[\text{SIZE_INTERVAL} \cdot i, \text{SIZE_INTERVAL} \cdot i + \text{SIZE_INTERVAL} - 1]$. C is the minimum number of values to change in the VARIABLES1 and VARIABLES2 collections so that for all integer i we have $M_i > 0 \Rightarrow N_i > 0$.
Arc input(s)	VARIABLES1 VARIABLES2
Arc generator	$\text{PRODUCT} \mapsto \text{collection}(\text{variables1}, \text{variables2})$
Arc arity	2
Arc constraint(s)	$\text{variables1.var}/\text{SIZE_INTERVAL} = \text{variables2.var}/\text{SIZE_INTERVAL}$
Graph property(ies)	$\text{NSINK_NSOURCE} = \text{VARIABLES2} - C$

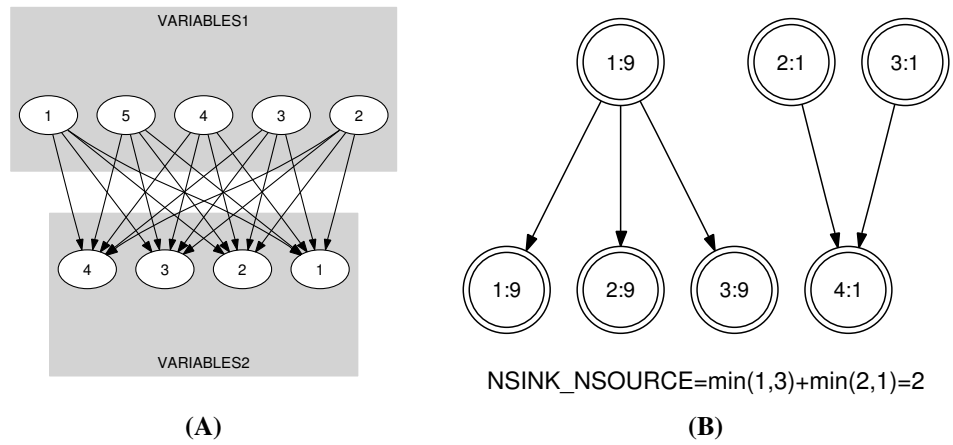
Example	$\text{soft_used_by_interval_var} \left(2, \left\{ \begin{array}{l} \text{var} - 9, \\ \text{var} - 1, \\ \text{var} - 1, \\ \text{var} - 8, \\ \text{var} - 8 \end{array} \right\}, \left\{ \begin{array}{l} \text{var} - 9, \\ \text{var} - 9, \\ \text{var} - 9, \\ \text{var} - 1 \end{array} \right\}, 3 \right)$
----------------	---

Parts (A) and (B) of Figure 4.387 respectively show the initial and final graph. Since we use the `NSINK_NSOURCE` graph property, the source and sink vertices of the final graph are stressed with a double circle. The `soft_used_by_interval_var` constraint holds since the cost 2 corresponds to the difference between the number of variables of VARIABLES2 and the sum over the different connected components of the minimum number of sources and sinks.

Usage A soft `used_by_interval` constraint.

See also `used_by_interval`.

Key words soft constraint, constraint between two collections of variables, relaxation, variable-based violation measure, interval.

Figure 4.387: Initial and final graph of the `soft_used_by_interval_var` constraint

20050507

831

4.199 `soft_used_by_modulo_var`

Origin	Derived from <code>used_by_modulo</code>
Constraint	<code>soft_used_by_modulo_var(C, VARIABLES1, VARIABLES2, M)</code>
Synonym(s)	<code>soft_used_by_modulo.</code>
Argument(s)	C : dvar $VARIABLES1$: collection(var - dvar) $VARIABLES2$: collection(var - dvar) M : int
Restriction(s)	$C \geq 0$ $C \leq VARIABLES2 $ $ VARIABLES1 \geq VARIABLES2 $ required(VARIABLES1, var) required(VARIABLES2, var) $M > 0$
Purpose	For each integer R in $[0, M - 1]$, let $N1_R$ (respectively $N2_R$) denote the number of variables of $VARIABLES1$ (respectively $VARIABLES2$) which have R as a rest when divided by M . C is the minimum number of values to change in the $VARIABLES1$ and $VARIABLES2$ collections so that for all R in $[0, M - 1]$ we have $N2_R > 0 \Rightarrow N1_R > 0$.
Arc input(s)	<code>VARIABLES1</code> <code>VARIABLES2</code>
Arc generator	$PRODUCT \mapsto \text{collection}(\text{variables1}, \text{variables2})$
Arc arity	2
Arc constraint(s)	<code>variables1.var mod M = variables2.var mod M</code>
Graph property(ies)	$NSINK_NSOURCE = VARIABLES2 - C$

Example	$\text{soft_used_by_modulo_var} \left(2, \left(\begin{array}{c} \text{var} - 9, \\ \text{var} - 1, \\ \text{var} - 1, \\ \text{var} - 8, \\ \text{var} - 8 \end{array} \right), \left(\begin{array}{c} \text{var} - 9, \\ \text{var} - 9, \\ \text{var} - 9, \\ \text{var} - 1 \end{array} \right), 3 \right)$
----------------	---

Parts (A) and (B) of Figure 4.388 respectively show the initial and final graph. Since we use the `NSINK_NSOURCE` graph property, the source and sink vertices of the final graph are stressed with a double circle. The `soft_used_by_modulo_var` constraint holds since the cost 2 corresponds to the difference between the number of variables of `VARIABLES2` and the sum over the different connected components of the minimum number of sources and sinks.

20050507

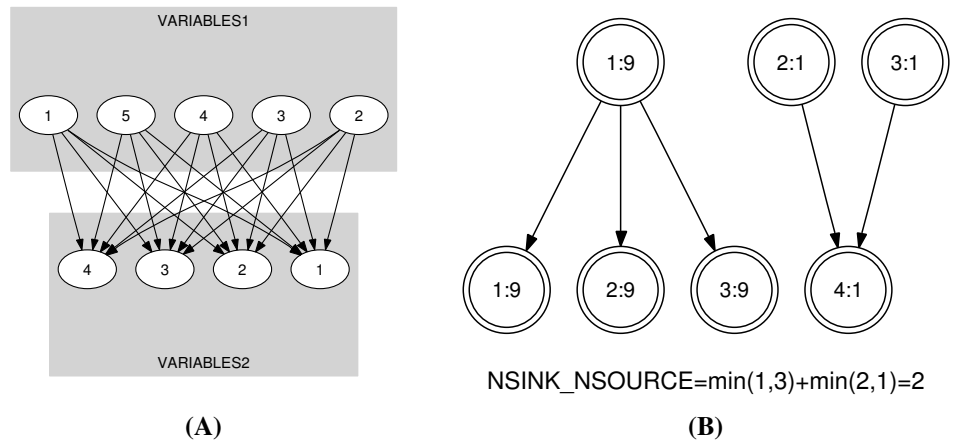
833

Usage

A soft `used_by_modulo` constraint.

Key words

soft constraint, constraint between two collections of variables, relaxation, variable-based violation measure, modulo.

Figure 4.388: Initial and final graph of the `soft_used_by_modulo_var` constraint

20050507

835

4.200 soft_used_by_partition_var

Origin	Derived from used_by_partition.
Constraint	soft_used_by_partition_var(C, VARIABLES1, VARIABLES2, PARTITIONS)
Synonym(s)	soft_used_by_partition.
Type(s)	VALUES : collection(val – int)
Argument(s)	C : dvar VARIABLES1 : collection(var – dvar) VARIABLES2 : collection(var – dvar) PARTITIONS : collection(p – VALUES)
Restriction(s)	$C \geq 0$ $C \leq \text{VARIABLES2} $ $ \text{VARIABLES1} \geq \text{VARIABLES2} $ required(VARIABLES1, var) required(VARIABLES2, var) required(PARTITIONS, p) $ \text{PARTITIONS} \geq 2$ required(VALUES, val) distinct(VALUES, val)
Purpose	For each integer i in $[1, \text{PARTITIONS}]$, let $N1_i$ (respectively $N2_i$) denote the number of variables of VARIABLES1 (respectively VARIABLES2) which take their value in the i^{th} partition of the collection PARTITIONS. C is the minimum number of values to change in the VARIABLES1 and VARIABLES2 collections so that for all i in $[1, \text{PARTITIONS}]$ we have $N1_i = N2_i$.
Arc input(s)	VARIABLES1 VARIABLES2
Arc generator	<i>PRODUCT</i> \mapsto collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	in_same_partition(variables1.var, variables2.var, PARTITIONS)
Graph property(ies)	<u>$\text{NSINK_NSOURCE} = \text{VARIABLES2} - C$</u>

Example

$$\text{soft_used_by_partition_var} \left(2, \left\{ \begin{array}{l} \text{var} - 9, \\ \text{var} - 1, \\ \text{var} - 1, \\ \text{var} - 8, \\ \text{var} - 8 \end{array} \right\}, \left\{ \begin{array}{l} \text{var} - 9, \\ \text{var} - 9, \\ \text{var} - 9, \\ \text{var} - 1 \end{array} \right\}, \left\{ \begin{array}{l} p - \{\text{val} - 1, \text{val} - 2\}, \\ p - \{\text{val} - 9\}, \\ p - \{\text{val} - 7, \text{val} - 8\} \end{array} \right\} \right)$$

Parts (A) and (B) of Figure 4.389 respectively show the initial and final graph. Since we use the `NSINK_NSOURCE` graph property, the source and sink vertices of the final graph are stressed with a double circle. The `soft_used_by_partition_var` constraint holds since the cost 2 corresponds to the difference between the number of variables of `VARIABLES2` and the sum over the different connected components of the minimum number of sources and sinks.

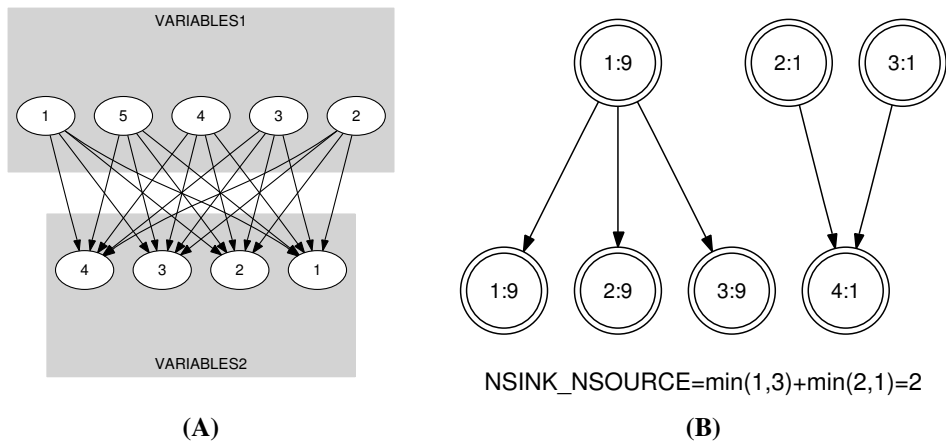


Figure 4.389: Initial and final graph of the `soft_used_by_partition_var` constraint

Usage A soft `used_by_partition` constraint.

See also `used_by_partition`.

Key words soft constraint, constraint between two collections of variables, relaxation, variable-based violation measure, partition.

4.201 `soft_used_by_var`

Origin	Derived from <code>used_by</code>
Constraint	<code>soft_used_by_var(C, VARIABLES1, VARIABLES2)</code>
Synonym(s)	<code>soft_used_by.</code>
Argument(s)	C : <code>dvar</code> $VARIABLES1$: <code>collection(var - dvar)</code> $VARIABLES2$: <code>collection(var - dvar)</code>
Restriction(s)	$C \geq 0$ $C \leq VARIABLES2 $ $ VARIABLES1 \geq VARIABLES2 $ <code>required(VARIABLES1, var)</code> <code>required(VARIABLES2, var)</code>
Purpose	<p>C is the minimum number of values to change in the <code>VARIABLES1</code> and <code>VARIABLES2</code> collections so that all the values of the variables of collection <code>VARIABLES2</code> are used by the variables of collection <code>VARIABLES1</code>.</p>
Arc input(s)	<code>VARIABLES1 VARIABLES2</code>
Arc generator	<code>PRODUCT</code> \mapsto <code>collection(variables1, variables2)</code>
Arc arity	2
Arc constraint(s)	<code>variables1.var = variables2.var</code>
Graph property(ies)	<code>NSINK_NSOURCE = VARIABLES2 - C</code>

Example	$\text{soft_used_by_var} \left(2, \left(\begin{array}{c} \left\{ \begin{array}{c} \text{var} - 9, \\ \text{var} - 1, \\ \text{var} - 1, \\ \text{var} - 8, \\ \text{var} - 8 \end{array} \right\}, \\ \left\{ \begin{array}{c} \text{var} - 9, \\ \text{var} - 9, \\ \text{var} - 9, \\ \text{var} - 1 \end{array} \right\} \end{array} \right) \right)$
----------------	---

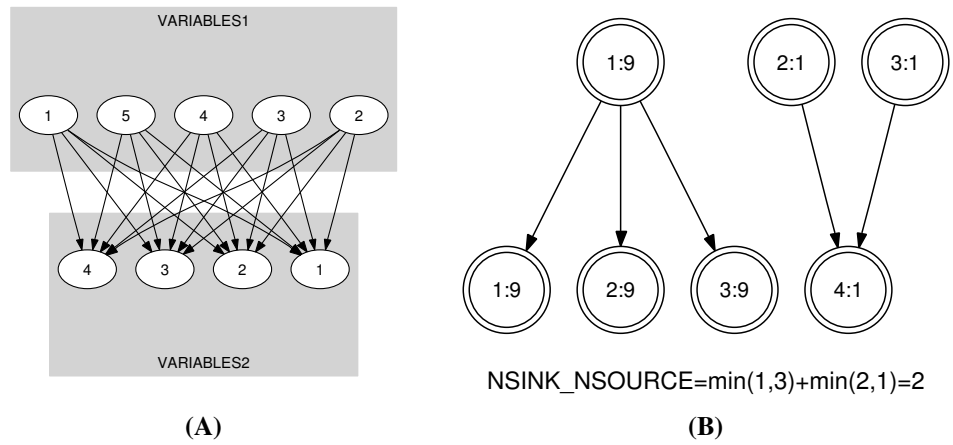
Parts (A) and (B) of Figure 4.390 respectively show the initial and final graph. Since we use the `NSINK_NSOURCE` graph property, the source and sink vertices of the final graph are stressed with a double circle. The `soft_used_by_var` constraint holds since the cost 2 corresponds to the difference between the number of variables of `VARIABLES2` and the sum over the different connected components of the minimum number of sources and sinks.

20050507

839

Usage A soft used_by constraint.

Key words soft constraint, constraint between two collections of variables, relaxation,
variable-based violation measure.

Figure 4.390: Initial and final graph of the `soft_used_by_var` constraint

20050507

841

4.202 sort

Origin	[139]
Constraint	sort(VARIABLES1, VARIABLES2)
Argument(s)	VARIABLES1 : collection(var – dvar) VARIABLES2 : collection(var – dvar)
Restriction(s)	VARIABLES1 = VARIABLES2 required(VARIABLES1, var) required(VARIABLES2, var)
Purpose	The variables of the collection VARIABLES2 correspond to the variables of VARIABLES1 according to a permutation. The variables of VARIABLES2 are also sorted in increasing order.
Arc input(s)	VARIABLES1 VARIABLES2
Arc generator	<i>PRODUCT</i> \mapsto collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	variables1.var = variables2.var
Graph property(ies)	<ul style="list-style-type: none"> • for all connected components: NSOURCE = NSINK • NSOURCE = VARIABLES1 • NSINK = VARIABLES2
Arc input(s)	VARIABLES2
Arc generator	<i>PATH</i> \mapsto collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	variables1.var \leq variables2.var
Graph property(ies)	NARC = VARIABLES2 – 1

Example

$$\text{sort} \left(\left(\left(\begin{array}{l} \text{var} - 1, \\ \text{var} - 9, \\ \text{var} - 1, \\ \text{var} - 5, \\ \text{var} - 2, \\ \text{var} - 1 \end{array} \right) \right), \left(\begin{array}{l} \text{var} - 1, \\ \text{var} - 1, \\ \text{var} - 1, \\ \text{var} - 2, \\ \text{var} - 5, \\ \text{var} - 9 \end{array} \right) \right)$$

Parts (A) and (B) of Figure 4.391 respectively show the initial and final graph associated to the first graph constraint. Since it uses the **NSOURCE** and **NSINK** graph properties, the source and sink vertices of this final graph are stressed with a double circle. Since there is a constraint on each connected component of the final graph we also show the different connected components. The **sort** constraint holds since:

- Each connected component of the final graph of the first graph constraint has the same number of sources and of sinks.
- The number of sources of the final graph of the first graph constraint is equal to $|\text{VARIABLES1}|$.
- The number of sinks of the final graph of the first graph constraint is equal to $|\text{VARIABLES2}|$.
- Finally the second graph constraint holds also since its corresponding final graph contains exactly $|\text{VARIABLES1} - 1|$ arcs: All the inequalities constraints between consecutive variables of **VARIABLES2** holds.

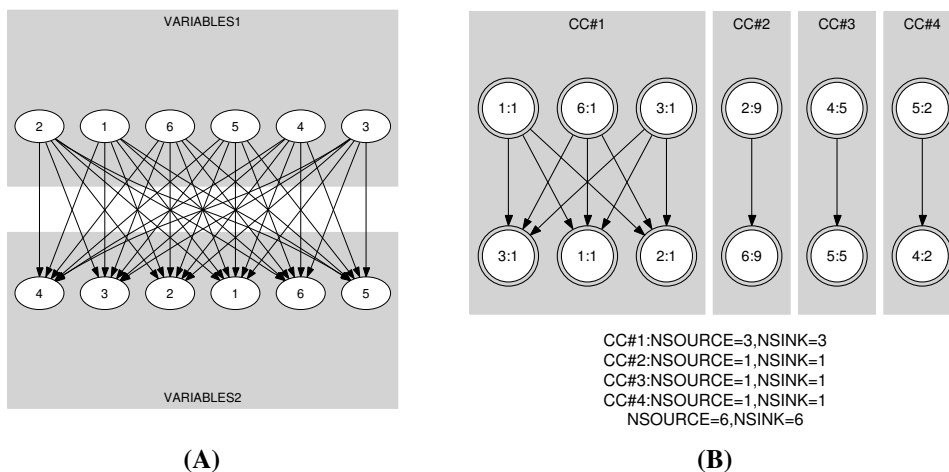


Figure 4.391: Initial and final graph of the **sort** constraint

Signature

Consider the first graph constraint. Since the initial graph contains only sources and sinks, and since isolated vertices are eliminated from the final graph, we make the following observations:

- Sources of the initial graph cannot become sinks of the final graph,
- Sinks of the initial graph cannot become sources of the final graph.

From the previous observations and since we use the *PRODUCT* arc generator on the collections **VARIABLES1** and **VARIABLES2**, we have that the maximum number of sources and sinks of the final graph is respectively equal to $|\text{VARIABLES1}|$ and $|\text{VARIABLES2}|$. Therefore we can rewrite $\text{NSOURCE} = |\text{VARIABLES1}|$ to $\text{NSOURCE} \geq |\text{VARIABLES1}|$ and simplify $\underline{\text{NSOURCE}}$ to NSOURCE . In a similar way, we can rewrite

844 $\overline{\text{NSINK}}, \overline{\text{NSOURCE}}, \text{CC}(\overline{\text{NSINK}}, \overline{\text{NSOURCE}}), \text{PRODUCT}; \overline{\text{NARC}}, \text{PATH}$

$\overline{\text{NSINK}} = |\text{VARIABLES2}|$ to $\text{NSINK} \geq |\text{VARIABLES2}|$ and simplify $\overline{\text{NSINK}}$ to NSINK .

Consider now the second graph constraint. Since we use the *PATH* arc generator with an arity of 2 on the *VARIABLES2* collection, the maximum number of arcs of the final graph is equal to $|\text{VARIABLES2}| - 1$. Therefore we can rewrite the graph property $\overline{\text{NARC}} = |\text{VARIABLES2}| - 1$ to $\text{NARC} \geq |\text{VARIABLES2}| - 1$ and simplify $\overline{\text{NARC}}$ to NARC .

Remark	A variant of this constraint was introduced in [147]. In this variant an additional list of domain variables represents the permutation which allows to go from <i>VARIABLES1</i> to <i>VARIABLES2</i> .
Algorithm	[61, 23].
See also	same, <i>sort_permutation</i> .
Key words	constraint between two collections of variables, sort, permutation.

20030820

845

4.203 `sort_permutation`

Origin	[147]
Constraint	<code>sort_permutation(FROM, PERMUTATION, TO)</code>
Usual name	<code>sort</code>
Argument(s)	<code>FROM</code> : <code>collection(var - dvar)</code> <code>PERMUTATION</code> : <code>collection(var - dvar)</code> <code>TO</code> : <code>collection(var - dvar)</code>
Restriction(s)	$ \text{PERMUTATION} = \text{FROM} $ $ \text{PERMUTATION} = \text{TO} $ $\text{PERMUTATION.var} \geq 1$ $\text{PERMUTATION.var} \leq \text{PERMUTATION} $ <code>alldifferent(PERMUTATION)</code> <code>required(FROM, var)</code> <code>required(PERMUTATION, var)</code> <code>required(TO, var)</code>
Purpose	The variables of collection FROM correspond to the variables of collection TO according to the permutation PERMUTATION. The variables of collection TO are also sorted in increasing order.
Derived Collection(s)	$\text{col} \left(\begin{array}{l} \text{FROM_PERMUTATION} - \text{collection}(\text{var} - \text{dvar}, \text{ind} - \text{dvar}), \\ [\text{item}(\text{var} - \text{FROM.var}, \text{ind} - \text{PERMUTATION.var})] \end{array} \right)$
Arc input(s)	<code>FROM_PERMUTATION TO</code>
Arc generator	$\text{PRODUCT} \mapsto \text{collection}(\text{from_permutation}, \text{to})$
Arc arity	2
Arc constraint(s)	<ul style="list-style-type: none"> • <code>from_permutation.var = to.var</code> • <code>from_permutation.ind = to.key</code>
Graph property(ies)	$\overline{\text{NARC}} = \text{PERMUTATION} $
Arc input(s)	<code>TO</code>
Arc generator	$\text{PATH} \mapsto \text{collection}(\text{to1}, \text{to2})$
Arc arity	2
Arc constraint(s)	<code>to1.var ≤ to2.var</code>
Graph property(ies)	$\overline{\text{NARC}} = \text{TO} - 1$

Example

$$\text{sort_permutation} \left(\left(\begin{array}{l} \text{var} - 1, \\ \text{var} - 9, \\ \text{var} - 1, \\ \text{var} - 5, \\ \text{var} - 2, \\ \text{var} - 1 \end{array} \right), \left(\begin{array}{l} \text{var} - 1, \\ \text{var} - 6, \\ \text{var} - 3, \\ \text{var} - 5, \\ \text{var} - 4, \\ \text{var} - 2 \end{array} \right), \left(\begin{array}{l} \text{var} - 1, \\ \text{var} - 1, \\ \text{var} - 1, \\ \text{var} - 2, \\ \text{var} - 5, \\ \text{var} - 9 \end{array} \right) \right)$$

Parts (A) and (B) of Figure 4.392 respectively show the initial and final graph associated to the first graph constraint. In both graphs the source vertices correspond to the items of the derived collection FROM_PERMUTATION, while the sink vertices correspond to the items of the TO collection. Since the first graph constraint uses the NARC graph property, the arcs of its final graph are stressed in bold. The sort_permutation constraint holds since:

- The first graph constraint holds since its final graph contains exactly PERMUTATION arcs.
- Finally the second graph constraint holds also since its corresponding final graph contains exactly |PERMUTATION - 1| arcs: All the inequalities constraints between consecutive variables of TO holds.

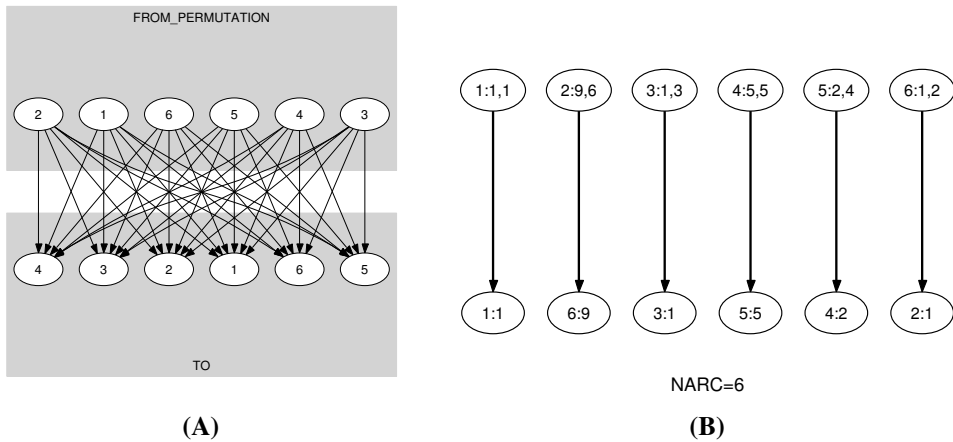


Figure 4.392: Initial and final graph of the sort_permutation constraint

Signature

Consider the first graph constraint where we use the *PRODUCT* arc generator. Since all the key attributes of the *T0* collection are distinct, and because of the second condition `from_permutation.ind = to.key` of the arc constraint, each vertex of the final graph has at most one successor. Therefore the maximum number of arcs of the final graph is equal to $|\text{PERMUTATION}|$. So we can rewrite the graph property $\text{NARC} = |\text{PERMUTATION}|$ to $\text{NARC} \geq |\text{PERMUTATION}|$ and simplify NARC to NARC.

Consider now the second graph constraint. Since we use the *PATH* arc generator with an arity of 2 on the *T0* collection, the maximum number of arcs of the corresponding final graph is equal to $|\text{T0}| - 1$. Therefore we can rewrite $\text{NARC} = |\text{T0}| - 1$ to $\text{NARC} \geq |\text{T0}| - 1$ and simplify NARC to NARC.

Algorithm

[147].

See also

correspondence, sort.

Key words

constraint between three collections of variables, sort, permutation, derived collection.

20030820

849

4.204 stage_element

Origin	CHOCO, derived from element.
Constraint	stage_element(<i>ITEM</i> , <i>TABLE</i>)
Usual name	stage_elt
Argument(s)	<i>ITEM</i> : collection(index – dvar, value – dvar) <i>TABLE</i> : collection(low – int, up – int, value – int)
Restriction(s)	required(<i>ITEM</i> , [index, value]) <i>ITEM</i> = 1 required(<i>TABLE</i> , [low, up, value])

Purpose

Let low_i , up_i and $value_i$ respectively denote the values of the low, up and value attributes of the i^{th} item of the *TABLE* collection. First we have that: $low_i \leq up_i$ and $up_i + 1 = low_{i+1}$. Second, the *stage_element* constraint enforces the following equivalence:
 $low_i \leq \text{ITEM.index} \wedge \text{ITEM.index} \leq up_i \Leftrightarrow \text{ITEM.value} = value_i$.

Arc input(s)	<i>TABLE</i>
Arc generator	<i>PATH</i> \mapsto collection(table1, table2)
Arc arity	2
Arc constraint(s)	<ul style="list-style-type: none"> • table1.low \leq table1.up • table1.up + 1 = table2.low • table2.low \leq table2.up
Graph property(ies)	$\text{NARC} = \text{TABLE} - 1$

Arc input(s)	<i>ITEM</i> <i>TABLE</i>
Arc generator	<i>PRODUCT</i> \mapsto collection(item, table)
Arc arity	2
Arc constraint(s)	<ul style="list-style-type: none"> • item.index \geq table.low • item.index \leq table.up • item.value = table.value
Graph property(ies)	$\text{NARC} = 1$

Example $\text{stage_element} \left(\begin{array}{c} \{ \text{index} - 5 \text{ value} - 6 \}, \\ \left\{ \begin{array}{ccc} \text{low} - 3 & \text{up} - 7 & \text{value} - 6, \\ \text{low} - 8 & \text{up} - 8 & \text{value} - 9, \\ \text{low} - 9 & \text{up} - 14 & \text{value} - 2, \\ \text{low} - 15 & \text{up} - 19 & \text{value} - 9 \end{array} \right\} \end{array} \right)$

Parts (A) and (B) of Figure 4.393 respectively show the initial and final graph associated to the second graph constraint. Since we use the **NARC** graph property, the unique arc of the final graph is stressed in bold.

Graph model The first graph constraint models the restrictions on the `low` and `up` attributes of the `TABLE` collection, while the second graph constraint is similar to the one used for defining the `element` constraint.

Automaton Figure 4.394 depicts the automaton associated to the `stage_element` constraint. Let `INDEX` and `VALUE` respectively be the `index` and the `value` attributes of the unique item of the `ITEM` collection. Let `LOWi`, `UPi` and `VALUEi` respectively be the `low`, the `up` and the `value` attributes of the *i*th item of the `TABLE` collection. To each quintuple (`INDEX`, `VALUE`, `LOWi`, `UPi`, `VALUEi`) corresponds a 0-1 signature variable `Si` as well as the following signature constraint: $((\text{LOW}_i \leq \text{INDEX}) \wedge (\text{INDEX} \leq \text{UP}_i) \wedge (\text{VALUE} = \text{VALUE}_i)) \Leftrightarrow S_i$.

See also `element`, `elem`.

Key words `data constraint`, `binary constraint`, `table`, `functional dependency`, `automaton`, `automaton without counters`, `centered cyclic(2) constraint network(1)`.

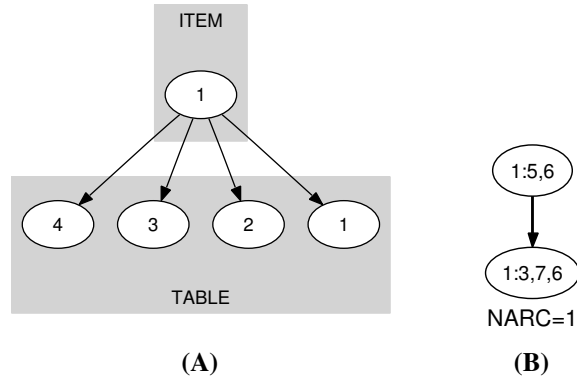


Figure 4.393: Initial and final graph of the `stage_element` constraint

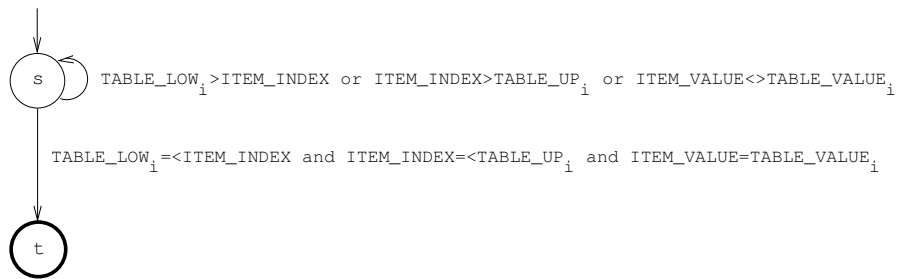


Figure 4.394: Automaton of the `stage_element` constraint

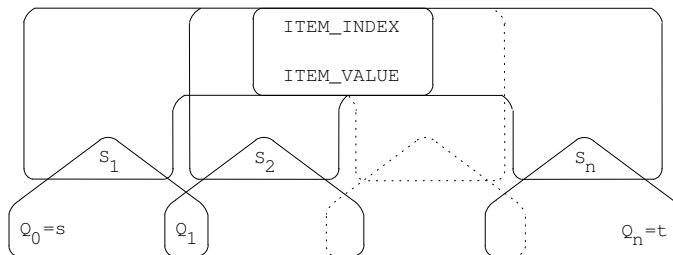


Figure 4.395: Hypergraph of the reformulation corresponding to the automaton of the `stage_element` constraint

20040828

853

4.205 stretch_circuit

Origin	[148]
Constraint	stretch_circuit(VARIABLES, VALUES)
Usual name	stretch
Argument(s)	VARIABLES : collection(var – dvar) VALUES : collection(val – int, lmin – int, lmax – int)
Restriction(s)	VARIABLES > 0 required(VARIABLES, var) VALUES > 0 required(VALUES, [val, lmin, lmax]) distinct(VALUES, val) VALUES.lmin ≤ VALUES.lmax

Purpose

Let n be the number of variables of the collection *VARIABLES*. Let X_i, \dots, X_j ($0 \leq i < n, 0 \leq j < n$) be consecutive variables of the collection of variables *VARIABLES* such that the following conditions apply:

- All variables X_i, \dots, X_j take a same value from the set of values of the *val* attribute,
- $X_{(i-1) \bmod n}$ is different from X_i ,
- $X_{(j+1) \bmod n}$ is different from X_j .

We call such a set of variables a *stretch*. The *span* of the stretch is equal to $1 + (j - i) \bmod n$, while the *value* of the stretch is X_i . An item (*val* – v , *lmin* – s , *lmax* – t) gives the minimum value s as well as the maximum value t for the span of a stretch of value v .

For all items of *VALUES*:

Arc input(s)	VARIABLES
Arc generator	<i>CIRCUIT</i> \mapsto collection(variables1, variables2) <i>LOOP</i> \mapsto collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	• variables1.var = VALUES.val • variables2.var = VALUES.val
Graph property(ies)	• not_in(MIN_NCC, 1, VALUES.lmin – 1) • MAX_NCC ≤ VALUES.lmax

Example

$$\text{stretch_circuit} \left(\left(\begin{array}{l} \text{var} - 6, \\ \text{var} - 6, \\ \text{var} - 3, \\ \text{var} - 1, \\ \text{var} - 1, \\ \text{var} - 1, \\ \text{var} - 6, \\ \text{var} - 6 \end{array} \right), \left(\begin{array}{lll} \text{val} - 1 & \text{lmin} - 2 & \text{lmax} - 4, \\ \text{val} - 2 & \text{lmin} - 2 & \text{lmax} - 3, \\ \text{val} - 3 & \text{lmin} - 1 & \text{lmax} - 6, \\ \text{val} - 6 & \text{lmin} - 2 & \text{lmax} - 4 \end{array} \right) \right)$$

Part (A) of Figure 4.396 shows the initial graphs associated to values 1, 2, 3 and 6. Part (B) of Figure 4.396 shows the final graphs associated to values 1, 3 and 6. Since value 2 is not assigned to any variable of the VARIABLES collection the final graph associated to value 2 is empty. The stretch_circuit constraint holds since:

- For value 1 we have one connected component for which the number of vertices is greater than or equal to 2 and less than or equal to 4,
- For value 2 we don't have any connected component,
- For value 3 we have one connected component for which the number of vertices is greater than or equal to 1 and less than or equal to 6,
- For value 6 we have one connected component for which the number of vertices is greater than or equal to 2 and less than or equal to 4.

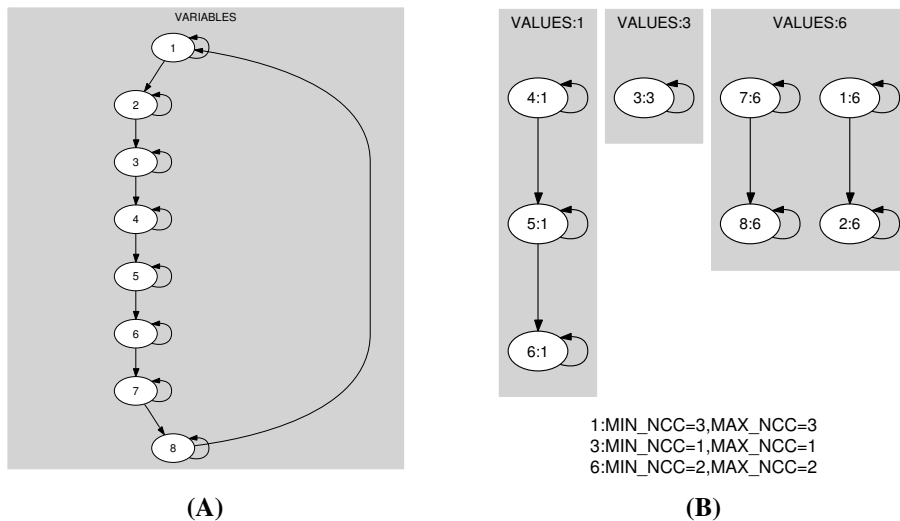


Figure 4.396: Initial and final graph of the stretch_circuit constraint

Usage

The paper [148] which originally introduced the stretch constraint quotes rostering problems as typical examples of use of this constraint.

- Remark** We split the origin `stretch` constraint into the `stretch_circuit` and the `stretch_path` constraints which respectively use the *PATH LOOP* and *CIRCUIT LOOP* arc generator. We also reorganize the parameters: the `VALUES` collection describes the attributes of each value that can be assigned to the variables of the `stretch_circuit` constraint. Finally we skipped the pattern constraint which tells what values can follow a given value.
- Algorithm** A first filtering algorithm was described in the original paper of G. Pesant [148]. An algorithm which also generates explanations is given in [7]. The first filtering algorithm achieving arc-consistency is depicted in [149]. This algorithm is based on dynamic programming and handles the fact that some values can be followed by only a given subset of values.
- See also** `stretch_path`, `sliding_distribution`, `group`, `pattern`.
- Key words** timetabling constraint, sliding sequence constraint, cyclic.

20030820

857

4.206 stretch_path

Origin	[148]
Constraint	stretch_path(VARIABLES, VALUES)
Usual name	stretch
Argument(s)	VARIABLES : collection(var – dvar) VALUES : collection(val – int, lmin – int, lmax – int)
Restriction(s)	VARIABLES > 0 required(VARIABLES, var) VALUES > 0 required(VALUES, [val, lmin, lmax]) distinct(VALUES, val) VALUES.lmin ≤ VALUES.lmax

Purpose

Let n be the number of variables of the collection VARIABLEs. Let X_i, \dots, X_j ($1 \leq i \leq j \leq n$) be consecutive variables of the collection of variables VARIABLEs such that the following conditions apply:

- All variables X_i, \dots, X_j take a same value from the set of values of the val attribute,
- $i = 1$ or X_{i-1} is different from X_i ,
- $j = n$ or X_{j+1} is different from X_j .

We call such a set of variables a *stretch*. The *span* of the stretch is equal to $j - i + 1$, while the *value* of the stretch is X_i . An item (val – v , lmin – s , lmax – t) gives the minimum value s as well as the maximum value t for the span of a stretch of value v .

For all items of VALUES:

Arc input(s)	VARIABLES
Arc generator	$PATH \mapsto \text{collection}(\text{variables1}, \text{variables2})$ $LOOP \mapsto \text{collection}(\text{variables1}, \text{variables2})$
Arc arity	2
Arc constraint(s)	• variables1.var = VALUES.val • variables2.var = VALUES.val
Graph property(ies)	• not_in(MIN_NCC, 1, VALUES.lmin – 1) • MAX_NCC ≤ VALUES.lmax

Example

$$\text{stretch_path} \left(\left(\left\{ \begin{array}{l} \text{var} - 6, \\ \text{var} - 6, \\ \text{var} - 3, \\ \text{var} - 1, \\ \text{var} - 1, \\ \text{var} - 1, \\ \text{var} - 6, \\ \text{var} - 6 \end{array} \right\}, \right. \right. \\ \left. \left. \left\{ \begin{array}{lll} \text{val} - 1 & \text{lmin} - 2 & \text{lmax} - 4, \\ \text{val} - 2 & \text{lmin} - 2 & \text{lmax} - 3, \\ \text{val} - 3 & \text{lmin} - 1 & \text{lmax} - 6, \\ \text{val} - 6 & \text{lmin} - 2 & \text{lmax} - 2 \end{array} \right\} \right) \right)$$

Part (A) of Figure 4.397 shows the initial graphs associated to values 1, 2, 3 and 6. Part (B) of Figure 4.397 shows the final graphs associated to values 1, 3 and 6. Since value 2 is not assigned to any variable of the VARIABLES collection the final graph associated to value 2 is empty. The `stretch_path` constraint holds since:

- For value 1 we have one connected component for which the number of vertices 3 is greater than or equal to 2 and less than or equal to 4,
- For value 2 we don't have any connected component,
- For value 3 we have one connected component for which the number of vertices 1 is greater than or equal to 1 and less than or equal to 6,
- For value 6 we have two connected components which both contain two vertices: This is greater than or equal to 2 and less than or equal to 2.

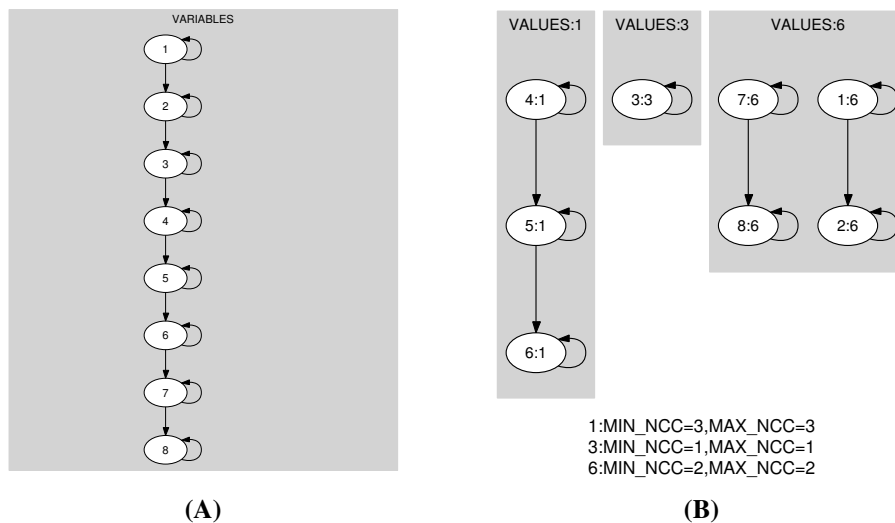


Figure 4.397: Initial and final graph of the `stretch_path` constraint

Graph model

During the presentation of this constraint at CP'2001 the following point was mentioned: It could be useful to allow domain variables for the minimum and the maximum values

of a stretch. This could be achieved in the following way: The `lmin` (respectively `lmax`) attribute would now be a domain variable which gives the size of the shortest (respectively longest) stretch. Finally within the `graph` property(ies) field we would replace \geq (and \leq) by $=$.

- Usage** The paper [148] which originally introduced the `stretch` constraint quotes rostering problems as typical examples of use of this constraint.
- Remark** We split the original `stretch` constraint into the `stretch_path` and the `stretch_circuit` constraints which respectively use the *PATH LOOP* and *CIRCUIT LOOP* arc generator. We also reorganize the parameters: the `VALUES` collection describes the attributes of each value that can be assigned to the variables of the `stretch_path` constraint. Finally we skipped the pattern constraint which tells what values can follow a given value.
- Algorithm** A first filtering algorithm was described in the original paper of G. Pesant [148]. A second filtering algorithm, based on dynamic programming, achieving arc-consistency is depicted in [149]. It also handles the fact that some values can be followed by only a given subset of values.
- See also** `stretch_circuit`, `sliding_distribution`, `group`, `pattern`.
- Key words** timetabling constraint, sliding sequence constraint.

20030820

861

4.207 `strict_lex2`

Origin	[123]
Constraint	<code>strict_lex2(MATRIX)</code>
Type(s)	<code>VECTOR</code> : <code>collection(var - dvar)</code>
Argument(s)	<code>MATRIX</code> : <code>collection(vec - VECTOR)</code>
Restriction(s)	<code>required(VECTOR, var)</code> <code>required(MATRIX, vec)</code> <code>same_size(MATRIX, vec)</code>
Purpose	Given a matrix of domain variables, enforces that both adjacent rows, and adjacent columns are lexicographically ordered (adjacent rows and adjacent columns cannot be equal).
Example	<code>strict_lex2</code> $\left(\left\{ \begin{array}{l} \text{vec} - \{\text{var} - 2, \text{var} - 2, \text{var} - 3\}, \\ \text{vec} - \{\text{var} - 2, \text{var} - 3, \text{var} - 1\} \end{array} \right\} \right)$
Usage	A <i>symmetry-breaking</i> constraint.
See also	<code>lex2</code> , <code>allperm</code> , <code>lex_lesseq</code> , <code>lex_chain_lesseq</code> .
Key words	predefined constraint, order constraint, matrix, matrix model, symmetry, lexicographic order.

4.208 `strictly_decreasing`

Origin	Derived from <code>strictly_increasing</code> .
Constraint	<code>strictly_decreasing(VARIABLES)</code>
Argument(s)	<code>VARIABLES : collection(var - dvar)</code>
Restriction(s)	<code> VARIABLES > 0</code> <code>required(VARIABLES, var)</code>
Purpose	The variables of the collection <code>VARIABLES</code> are strictly decreasing.
<hr/>	
Arc input(s)	<code>VARIABLES</code>
Arc generator	<code>PATH</code> \mapsto <code>collection(variables1, variables2)</code>
Arc arity	2
Arc constraint(s)	<code>variables1.var > variables2.var</code>
Graph property(ies)	$\text{NARC} = \text{VARIABLES} - 1$
<hr/>	
Example	$\text{strictly_decreasing} \left(\left(\begin{array}{c} \text{var} - 8, \\ \text{var} - 4, \\ \text{var} - 3, \\ \text{var} - 1 \end{array} \right) \right)$ <p>Parts (A) and (B) of Figure 4.398 respectively show the initial and final graph. Since we use the NARC graph property, the arcs of the final graph are stressed in bold.</p>
Automaton	Figure 4.399 depicts the automaton associated to the <code>strictly_decreasing</code> constraint. To each pair of consecutive variables ($\text{VAR}_i, \text{VAR}_{i+1}$) of the collection <code>VARIABLES</code> corresponds a 0-1 signature variable S_i . The following signature constraint links $\text{VAR}_i, \text{VAR}_{i+1}$ and S_i : $\text{VAR}_i \leq \text{VAR}_{i+1} \Leftrightarrow S_i$.
See also	<code>decreasing</code> , <code>increasing</code> , <code>strictly_increasing</code> .
Key words	decomposition, order constraint, automaton, automaton without counters, sliding cyclic(1) constraint network(1).

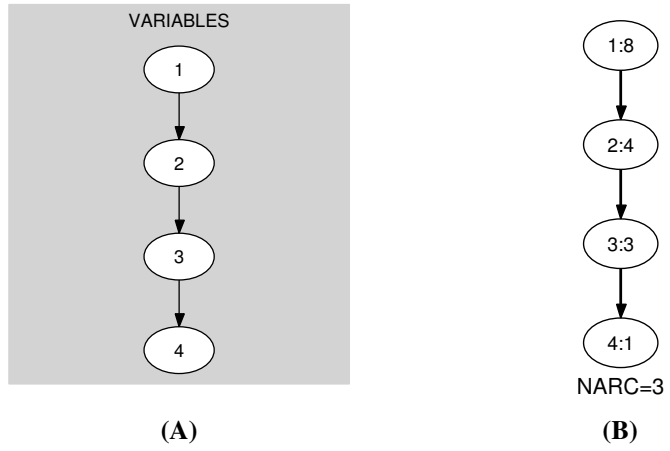


Figure 4.398: Initial and final graph of the `strictly_decreasing` constraint

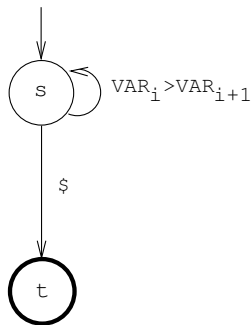


Figure 4.399: Automaton of the `strictly_decreasing` constraint

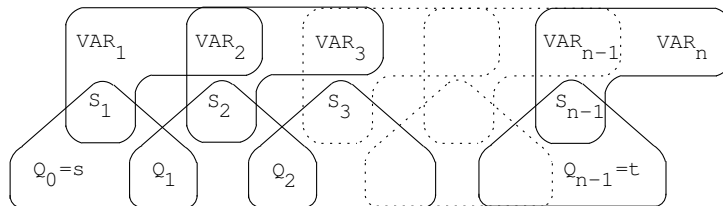


Figure 4.400: Hypergraph of the reformulation corresponding to the automaton of the `strictly_decreasing` constraint

4.209 `strictly_increasing`

Origin	KOALOG
Constraint	<code>strictly_increasing(VARIABLES)</code>
Argument(s)	<code>VARIABLES : collection(var – dvar)</code>
Restriction(s)	<code> VARIABLES > 0</code> <code>required(VARIABLES, var)</code>
Purpose	The variables of the collection <code>VARIABLES</code> are strictly increasing.
<hr/>	
Arc input(s)	<code>VARIABLES</code>
Arc generator	<code>PATH</code> \mapsto <code>collection(variables1, variables2)</code>
Arc arity	2
Arc constraint(s)	<code>variables1.var < variables2.var</code>
Graph property(ies)	$\text{NARC} = \text{VARIABLES} - 1$
<hr/>	
Example	$\text{strictly_increasing} \left(\left(\begin{array}{c} \text{var} - 1, \\ \text{var} - 3, \\ \text{var} - 4, \\ \text{var} - 8 \end{array} \right) \right)$ <p>Parts (A) and (B) of Figure 4.401 respectively show the initial and final graph. Since we use the NARC graph property, the arcs of the final graph are stressed in bold.</p>
Automaton	Figure 4.402 depicts the automaton associated to the <code>strictly_increasing</code> constraint. To each pair of consecutive variables ($\text{VAR}_i, \text{VAR}_{i+1}$) of the collection <code>VARIABLES</code> corresponds a 0-1 signature variable S_i . The following signature constraint links $\text{VAR}_i, \text{VAR}_{i+1}$ and S_i : $\text{VAR}_i \geq \text{VAR}_{i+1} \Leftrightarrow S_i$.
See also	<code>increasing</code> , <code>decreasing</code> , <code>strictly_decreasing</code> .
Key words	decomposition, order constraint, automaton, automaton without counters, sliding cyclic(1) constraint network(1).

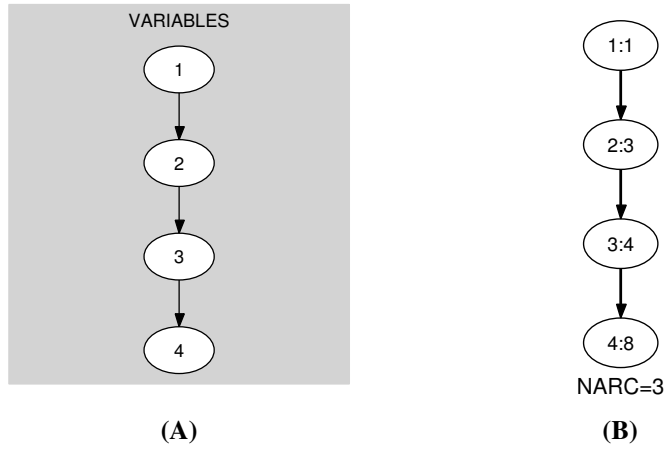


Figure 4.401: Initial and final graph of the `strictly_increasing` constraint

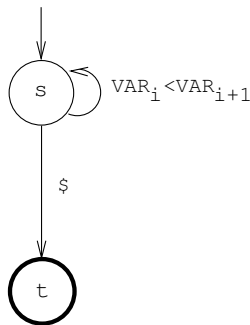


Figure 4.402: Automaton of the `strictly_increasing` constraint

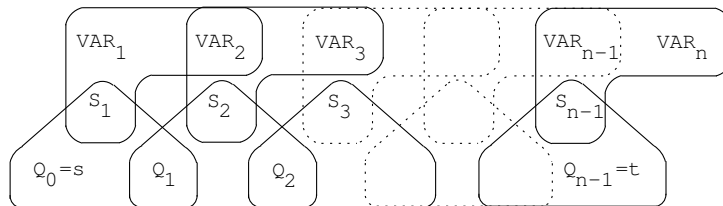


Figure 4.403: Hypergraph of the reformulation corresponding to the automaton of the `strictly_increasing` constraint

4.210 strongly_connected

Origin	[74]
Constraint	<code>strongly_connected(NODES)</code>
Argument(s)	<code>NODES : collection(index – int, succ – svar)</code>
Restriction(s)	<code>required(NODES, [index, succ])</code> <code>NODES.index ≥ 1</code> <code>NODES.index ≤ NODES </code> <code>distinct(NODES, index)</code>
Purpose	Consider a digraph G described by the <code>NODES</code> collection. Select a subset of arcs of G so that we have one single strongly connected component involving all vertices of G .
<hr/>	
Arc input(s)	<code>NODES</code>
Arc generator	<code>CLIQUE ↦ collection(nodes1, nodes2)</code>
Arc arity	2
Arc constraint(s)	<code>in_set(nodes2.index, nodes1.succ)</code>
Graph property(ies)	<code>MIN_NSCC = NODES </code>
<hr/>	
Example	$\text{strongly_connected} \left(\left\{ \begin{array}{l} \text{index} - 1 \quad \text{succ} - \{2\}, \\ \text{index} - 2 \quad \text{succ} - \{3\}, \\ \text{index} - 3 \quad \text{succ} - \{2, 5\}, \\ \text{index} - 4 \quad \text{succ} - \{1\}, \\ \text{index} - 5 \quad \text{succ} - \{4\} \end{array} \right\} \right)$ <p>Part (A) of Figure 4.404 shows the initial graph from which we start. It is derived from the set associated to each vertex. Each set describes the potential values of the <code>succ</code> attribute of a given vertex. Part (B) of Figure 4.404 gives the final graph associated to the example. The <code>strongly_connected</code> constraint holds since the final graph contains one single strongly connected component mentioning every vertex of the initial graph.</p>
Signature	Since the maximum number of vertices of the final graph is equal to $ NODES $ we can rewrite the graph property $\text{MIN_NSCC} = NODES $ to $\text{MIN_NSCC} \geq NODES $ and simplify <u>MIN_NSCC</u> to <u>MIN_NSCC</u> .
See also	<code>circuit</code> , <code>link_set_to_booleans</code> .
Key words	graph constraint, linear programming, strongly connected component, constraint involving set variables.

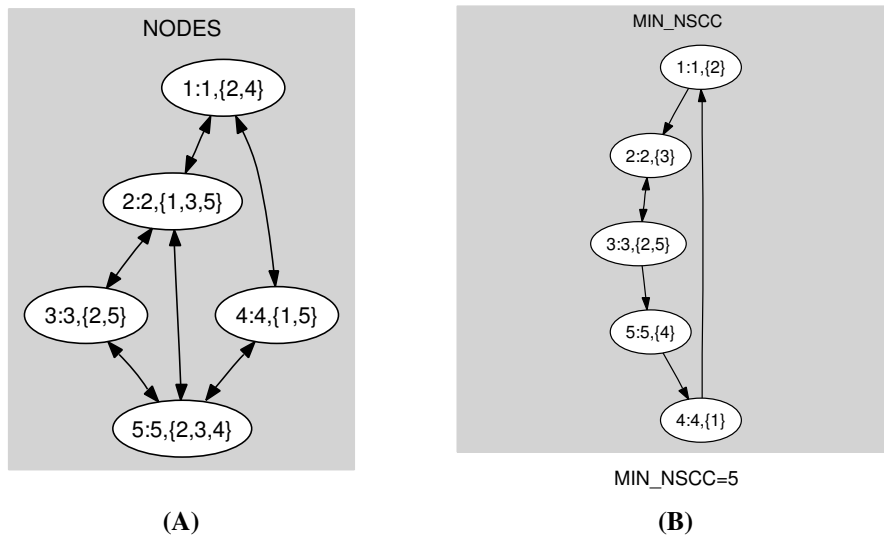


Figure 4.404: Initial and final graph of the strongly_connected set constraint

4.211 sum

Origin	[150].
Constraint	sum(INDEX, SETS, CONSTANTS, S)
Argument(s)	INDEX : dvar SETS : collection(ind - int, set - sint) CONSTANTS : collection(cst - int) S : dvar
Restriction(s)	$ SETS \geq 1$ required(SETS, [ind, set]) distinct(SETS, ind) $ CONSTANTS \geq 1$ required(CONSTANTS, cst)
Purpose	S is equal to the sum of the constants corresponding to the INDEX th set of the SETS collection.
Arc input(s)	SETS CONSTANTS
Arc generator	PRODUCT \mapsto collection(sets, constants)
Arc arity	2
Arc constraint(s)	<ul style="list-style-type: none"> • INDEX = sets.ind • in_set(constants.key, sets.set)
Graph property(ies)	SUM(CONSTANTS, cst) = S

Example

$$\text{sum} \left(\left(8, \left\{ \begin{array}{ll} \text{ind} - 8 & \text{set} - \{2, 3\}, \\ \text{ind} - 1 & \text{set} - \{3\}, \\ \text{ind} - 3 & \text{set} - \{1, 4, 5\}, \\ \text{ind} - 6 & \text{set} - \{2, 4\} \end{array} \right\} \right), \left(\begin{array}{l} \text{cst} - 4, \\ \text{cst} - 9, \\ \text{cst} - 1, \\ \text{cst} - 3, \\ \text{cst} - 1 \end{array} \right), 10 \right)$$

Parts (A) and (B) of Figure 4.405 respectively show the initial and final graph. Since we use the **SUM** graph property we show the vertices from which we compute S in a box.

Graph model According to the value assigned to INDEX the arc constraint selects for the final graph:

- The INDEXth item of the SETS collection,
- The items of the CONSTANTS collection for which the key correspond to the indices of the INDEXth set of the SETS collection.

Finally, since we use the **SUM** graph property on the `cst` attribute of the **CONSTANTS** collection, the last argument `S` of the `sum` constraint is equal to the sum of the constants associated to the vertices of the final graph.

- Usage** In his paper introducing the `sum` constraint, Tallys H. Yunes mentions the *Sequence Dependent Cumulative Cost Problem* as the subproblem that originally motivate this constraint.
- Algorithm** The paper [150] gives the convex hull relaxation of the `sum` constraint.
- See also** `element`, `sum_ctr`, `sum_set`.
- Key words** data constraint, linear programming, convex hull relaxation, `sum`.

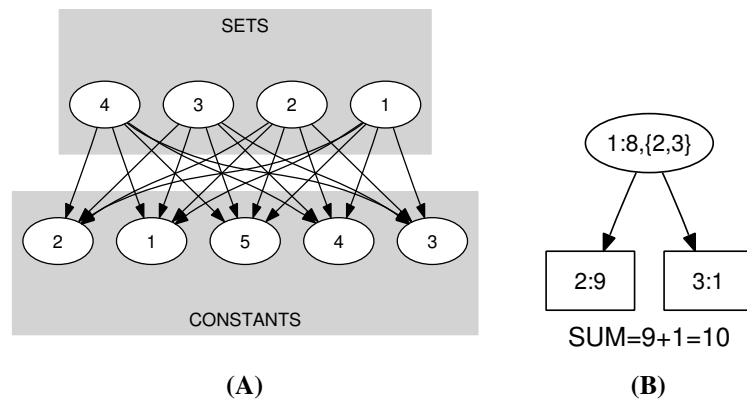


Figure 4.405: Initial and final graph of the sum constraint

20030820

873

4.212 sum_ctr

Origin	Arithmetic constraint.
Constraint	<code>sum_ctr(VARIABLES, CTR, VAR)</code>
Synonym(s)	<code>constant_sum</code> .
Argument(s)	<p><code>VARIABLES</code> : <code>collection(var - dvar)</code> <code>CTR</code> : <code>atom</code> <code>VAR</code> : <code>dvar</code></p>
Restriction(s)	<p><code>required(VARIABLES, var)</code> <code>CTR</code> \in $[=, \neq, <, \geq, >, \leq]$</p>
Purpose	<p>Constraint the sum of a set of domain variables. More precisely let S denotes the sum of the variables of the <code>VARIABLES</code> collection. Enforce the following constraint to hold: S <code>CTR</code> <code>VAR</code>.</p>
Arc input(s)	<code>VARIABLES</code>
Arc generator	<i>SELF</i> \mapsto <code>collection(variables)</code>
Arc arity	1
Arc constraint(s)	<code>TRUE</code>
Graph property(ies)	<code>SUM(VARIABLES, var) CTR VAR</code>
Example	<code>sum_ctr({var - 1, var - 1, var - 4}, =, 6)</code>

Parts (A) and (B) of Figure 4.406 respectively show the initial and final graph. Since we use the `TRUE` arc constraint both graphs are identical.

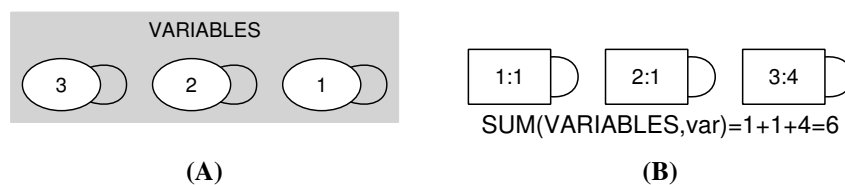


Figure 4.406: Initial and final graph of the `sum_ctr` constraint

Graph model	Since we want to keep all the vertices of the initial graph we use the <i>SELF</i> arc generator together with the <code>TRUE</code> arc constraint. This predefined arc constraint always holds.
Remark	When <code>CTR</code> corresponds to <code>=</code> this constraint is referenced under the name <code>constant_sum</code> in <code>KOALOG</code> .

Used in bin_packing, cumulative, cumulative_two_d,
cumulative_with_level_of_priority, cumulatives, indexed_sum,
interval_and_sum, relaxed_sliding_sum, sliding_sum,
sliding_time_window_sum.

See also sum, sum_set, product_ctr, range_ctr.

Key words arithmetic constraint, sum.

4.213 sum_of_weights_of_distinct_values

Origin	[106]
Constraint	sum_of_weights_of_distinct_values(VARIABLES, VALUES, COST)
Synonym(s)	swdv.
Argument(s)	VARIABLES : collection(var – dvar) VALUES : collection(val – int, weight – int) COST : dvar
Restriction(s)	required(VARIABLES, var) required(VALUES, [val, weight]) VALUES.weight ≥ 0 distinct(VALUES, val) COST ≥ 0
Purpose	All variables of the VARIABLE collection take a value in the VALUE collection. In addition COST is the sum of the weight attributes associated to the distinct values taken by the variables of VARIABLE.
Arc input(s)	VARIABLES VALUES
Arc generator	<i>PRODUCT</i> ↦ collection(variables, values)
Arc arity	2
Arc constraint(s)	variables.var = values.val
Graph property(ies)	<ul style="list-style-type: none"> • NSOURCE = VARIABLES • SUM(VALUES, weight) = COST

Example	$\text{sum_of_weights_of_distinct_values} \left(\left(\left\{ \begin{array}{l} \text{var} - 1, \\ \text{var} - 6, \\ \text{var} - 1 \end{array} \right\}, \left\{ \begin{array}{ll} \text{val} - 1 & \text{weight} - 5, \\ \text{val} - 2 & \text{weight} - 3, \\ \text{val} - 6 & \text{weight} - 7 \end{array} \right\}, 12 \right) \right)$
----------------	---

Parts (A) and (B) of Figure 4.407 respectively show the initial and final graph. Since we use the **NSOURCE** graph property, the source vertices of the final graph are shown in a double circle. Since we also use the **SUM** graph property we show the vertices from which we compute the total cost in a box.

Signature	Since we use the <i>PRODUCT</i> arc generator, the number of sources of the final graph cannot exceed the number of sources of the initial graph. Since the initial graph contains VARIABLES sources, this number is an upper bound of the number of sources of the final graph. Therefore we can rewrite NSOURCE = VARIABLES to NSOURCE ≥ VARIABLES and simplify <u>NSOURCE</u> to NSOURCE .
------------------	---

See also `minimum_weight_alldifferent`, `global_cardinality_with_costs`, `nvalue`,
 `weighted_partial_alldiff`.

Key words cost filtering constraint, assignment, relaxation, domination, weighted assignment,
 facilities location problem.

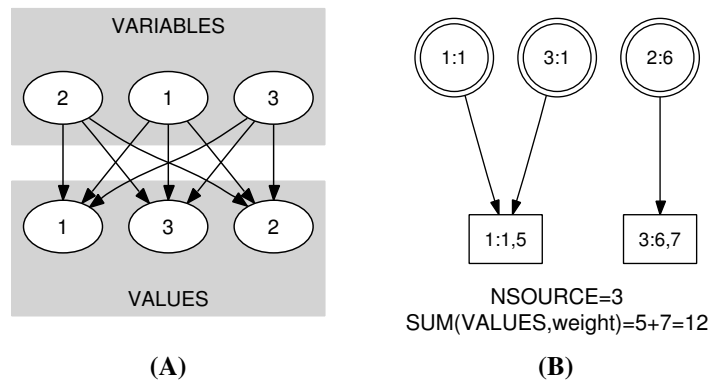


Figure 4.407: Initial and final graph of the `sum_of_weights_of_distinct_values` constraint

20030820

879

4.214 `sum_set`

Origin	H. Cambazard
Constraint	<code>sum_set(SV, VALUES, CTR, VAR)</code>
Argument(s)	<code>SV</code> : svar <code>VALUES</code> : collection(val - int, coef - int) <code>CTR</code> : atom <code>VAR</code> : dvar
Restriction(s)	<code>required(VALUES, [val, coef])</code> <code>distinct(VALUES, val)</code> <code>VALUES.coef ≥ 0</code> <code>CTR ∈ [=, ≠, <, ≥, >, ≤]</code>
Purpose	Let SUM denotes the sum of the coef attributes of the VALUES collection for which the corresponding values val occur in the set SV. Enforce the following constraint to hold: SUM CTR VAR.
Arc input(s)	VALUES
Arc generator	<code>SELF</code> \mapsto collection(values)
Arc arity	1
Arc constraint(s)	<code>in_set(values.val, SV)</code>
Graph property(ies)	<code>SUM(VALUES, coef) CTR VAR</code>

Example

$$\text{sum_set} \left(\left(\begin{array}{l} \{2, 3, 6\}, \\ \left\{ \begin{array}{ll} \text{val} - 2 & \text{coef} - 7, \\ \text{val} - 9 & \text{coef} - 1, \\ \text{val} - 5 & \text{coef} - 7, \\ \text{val} - 6 & \text{coef} - 2 \end{array} \right\}, \text{=}, 9 \end{array} \right) \right)$$

Parts (A) and (B) of Figure 4.408 respectively show the initial and final graph.

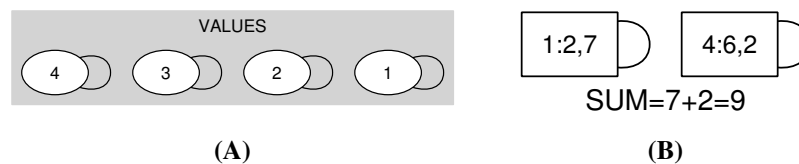


Figure 4.408: Initial and final graph of the `sum_set` constraint

See also `sum`, `sum_ctr`.

Key words arithmetic constraint, binary constraint, sum, constraint involving set variables.

20031001

881

4.215 symmetric_alldifferent

Origin	[20]
Constraint	symmetric_alldifferent(NODES)
Synonym(s)	symmetric_alldiff, symmetric_alldistinct, symm_alldifferent, symm_alldiff, symm_alldistinct, one_factor.
Argument(s)	NODES : collection(index – int, succ – dvar)
Restriction(s)	required(NODES, [index, succ]) NODES.index ≥ 1 NODES.index $\leq \text{NODES} $ distinct(NODES, index) NODES.succ ≥ 1 NODES.succ $\leq \text{NODES} $
Purpose	All variables associated to the succ attribute of the NODES collection should be pairwise distinct. In addition enforce the following condition: If variable NODES[i].succ takes value j then variable NODES[j].succ takes value i . This can be interpreted as a graph-covering problem where one has to cover a digraph G with circuits of length two in such a way that each vertex of G belongs to one single circuit.
Arc input(s)	NODES
Arc generator	<i>CLIQUE</i> (\neq) \mapsto collection(nodes1, nodes2)
Arc arity	2
Arc constraint(s)	<ul style="list-style-type: none"> • nodes1.succ = nodes2.index • nodes2.succ = nodes1.index
Graph property(ies)	<u>NARC</u> = NODES

Example symmetric_alldifferent $\left(\left(\begin{array}{l} \text{index} - 1 \quad \text{succ} - 3, \\ \text{index} - 2 \quad \text{succ} - 4, \\ \text{index} - 3 \quad \text{succ} - 1, \\ \text{index} - 4 \quad \text{succ} - 2 \end{array} \right) \right)$

Parts (A) and (B) of Figure 4.409 respectively show the initial and final graph. Since we use the NARC graph property, the arcs of the final graph are stressed in bold.

Graph model In order to express the binary constraint that links two vertices one has to make explicit the identifier of the vertices.

Signature Since all the index attributes of the NODES collection are distinct, and because of the first condition nodes1.succ = nodes2.index of the arc constraint, each vertex of the final graph has at most one successor. Therefore the maximum number of arcs of the final graph is equal to the maximum number of vertices |NODES| of the final graph. So we can rewrite NARC = |NODES| to NARC \geq |NODES| and simplify NARC to NARC.

- Usage** As it was reported in [20, page 420], this constraint is useful to express matches between persons. The `symmetric_alldifferent` constraint also appears implicitly in the *cycle cover problem* and corresponds to the four conditions given in section 1 *Modeling the Cycle Cover Problem* of [151].
- Remark** This constraint is referenced under the name `one_factor` in [152] as well as in [153]. From a modelling point of view this constraint can be express with the `cycle` constraint [37] where one imposes the additional condition that each cycle has only two nodes.
- Algorithm** [20].
- See also** `cycle`, `alldifferent`.
- Key words** graph constraint, circuit, cycle, timetabling constraint, sport timetabling, permutation, all different, disequality, graph partitioning constraint, matching.

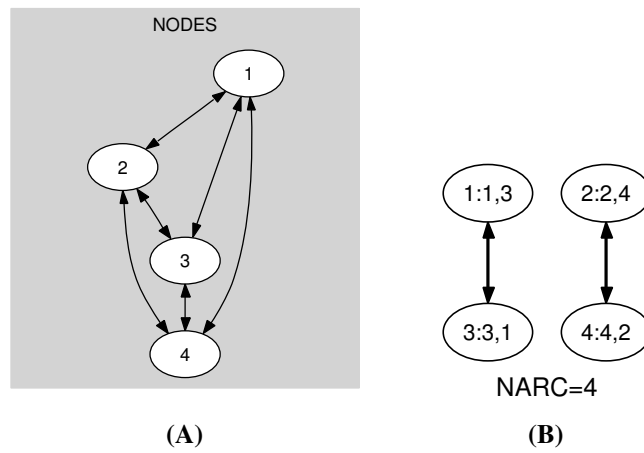


Figure 4.409: Initial and final graph of the symmetric_alldifferent constraint

20000128

885

4.216 symmetric_cardinality

Origin	Derived from <code>global_cardinality</code> by W. Kocjan.
Constraint	<code>symmetric_cardinality(VARS, VALS)</code>
Argument(s)	<p><code>VARS</code> : <code>collection(idvar – int, var – svar, l – int, u – int)</code> <code>VALS</code> : <code>collection(idval – int, val – svar, l – int, u – int)</code></p>
Restriction(s)	<pre> required(VARS, [idvar, var, l, u]) VARS ≥ 1 VARS.idvar ≥ 1 VARS.idvar ≤ VARS distinct(VARS, idvar) VARS.l ≥ 0 VARS.l ≤ VARS.u VARS.u ≤ VALS required(VALS, [idval, val, l, u]) VALS ≥ 1 VALS.idval ≥ 1 VALS.idval ≤ VALS distinct(VALS, idval) VALS.l ≥ 0 VALS.l ≤ VALS.u VALS.u ≤ VARS </pre>
Purpose	Put in relation two sets: For each element of one set gives the corresponding elements of the other set to which it is associated. In addition, it constraints the number of elements associated to each element to be in a given interval.
<hr/>	
Arc input(s)	<code>VARS VALS</code>
Arc generator	<code>PRODUCT</code> \mapsto <code>collection(vars, vals)</code>
Arc arity	2
Arc constraint(s)	<ul style="list-style-type: none"> • <code>in_set(vars.idvar, vals.val) \Leftrightarrow in_set(vals.idval, vars.var)</code> • <code>vars.l ≤ card_set(vars.var)</code> • <code>vars.u ≥ card_set(vars.var)</code> • <code>vals.l ≤ card_set(vals.val)</code> • <code>vals.u ≥ card_set(vals.val)</code>
Graph property(ies)	<code>NARC = VARS * VALS </code>

Example

$$\text{symmetric_cardinality} \left(\begin{array}{l} \left\{ \begin{array}{l} \text{idvar} - 1 \quad \text{var} - \{3\} \quad 1 - 0 \quad \text{u} - 1, \\ \text{idvar} - 2 \quad \text{var} - \{1\} \quad 1 - 1 \quad \text{u} - 2, \\ \text{idvar} - 3 \quad \text{var} - \{1, 2\} \quad 1 - 1 \quad \text{u} - 2, \\ \text{idvar} - 4 \quad \text{var} - \{1, 3\} \quad 1 - 2 \quad \text{u} - 3 \end{array} \right\}, \\ \left\{ \begin{array}{l} \text{idval} - 1 \quad \text{val} - \{2, 3, 4\} \quad 1 - 3 \quad \text{u} - 4, \\ \text{idval} - 2 \quad \text{val} - \{3\} \quad 1 - 1 \quad \text{u} - 1, \\ \text{idval} - 3 \quad \text{val} - \{1, 4\} \quad 1 - 1 \quad \text{u} - 2, \\ \text{idval} - 4 \quad \text{val} - \emptyset \quad 1 - 0 \quad \text{u} - 1 \end{array} \right\} \end{array} \right)$$

Parts (A) and (B) of Figure 4.410 respectively show the initial and final graph. Since we use the **NARC** graph property, all the arcs of the final graph are stressed in bold.

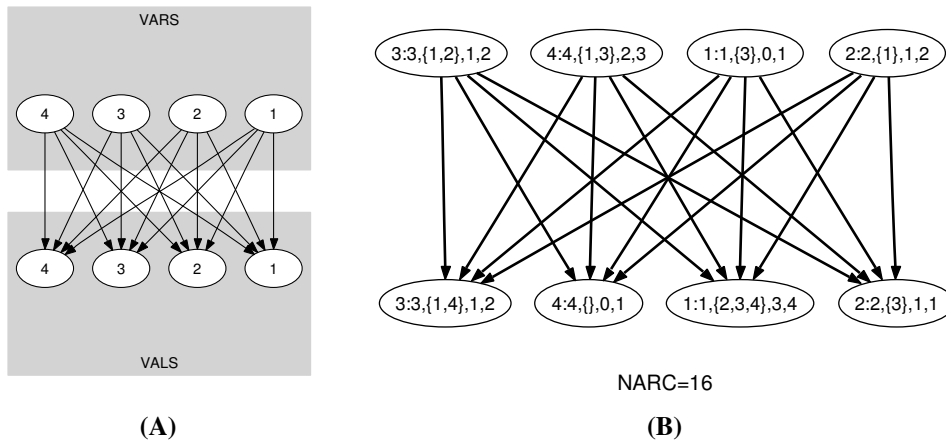


Figure 4.410: Initial and final graph of the `symmetric_cardinality` constraint

Graph model

The graph model used for the `symmetric_cardinality` is similar to the one used in the `domain_constraint` or in the `link_set_to_booleans` constraints: We use an equivalence in the arc constraint and ask all arc constraints to hold.

Signature

Since we use the *PRODUCT* arc generator on the collections `VARs` and `VALs`, the number of arcs of the initial graph is equal to $|\text{VARs}| \cdot |\text{VALs}|$. Therefore the maximum number of arcs of the final graph is also equal to $|\text{VARs}| \cdot |\text{VALs}|$ and we can rewrite $\text{NARC} = |\text{VARs}| \cdot |\text{VALs}|$ to $\text{NARC} \geq |\text{VARs}| \cdot |\text{VALs}|$. So we can simplify NARC to $\overline{\text{NARC}}$.

Usage

The most simple example of applying `symmetric_gcc` is a variant of personnel assignment problem, where one person can be assigned to perform between n and m ($n \leq m$) jobs, and every job requires between p and q ($p \leq q$) persons. In addition every job requires different kind of skills. The previous problem can be modelled as follows:

- For each person we create an item of the `VARs` collection,
- For each job we create an item of the `VALs` collection,
- There is an arc between a person and the particular job if this person is qualified to perform it.

- Remark** The `symmetric_gcc` constraint generalizes the `global_cardinality` constraint by allowing a variable to take more than one value.
- Algorithm** A flow-based arc-consistency algorithm for the `symmetric_cardinality` constraint is described in [154].
- See also** `symmetric_gcc`, `global_cardinality`, `link_set_to_booleans`.
- Key words** decomposition, timetabling constraint, assignment, relation, flow, constraint involving set variables.

20040530

889

4.217 symmetric_gcc

Origin	Derived from <code>global_cardinality</code> by W. Kocjan.
Constraint	<code>symmetric_gcc(VARS, VALS)</code>
Synonym(s)	<code>sgcc</code> .
Argument(s)	<p><code>VARS</code> : <code>collection(idvar - int, var - svar, nocc - dvar)</code> <code>VALS</code> : <code>collection(idval - int, val - svar, nocc - dvar)</code></p>

Restriction(s)	<pre> required(VARS, [idvar, var, nocc]) VARS ≥ 1 VARS.idvar ≥ 1 VARS.idvar ≤ VARS distinct(VARS, idvar) VARS.nocc ≥ 0 VARS.nocc ≤ VALS required(VALS, [idval, val, nocc]) VALS ≥ 1 VALS.idval ≥ 1 VALS.idval ≤ VALS distinct(VALS, idval) VALS.nocc ≥ 0 VALS.nocc ≤ VARS </pre>
-----------------------	--

Purpose	Put in relation two sets: For each element of one set gives the corresponding elements of the other set to which it is associated. In addition, enforce a cardinality constraint on the number of occurrences of each value.
----------------	--

Arc input(s)	<code>VARS VALS</code>
Arc generator	<code>PRODUCT</code> \mapsto <code>collection(vars, vals)</code>
Arc arity	2
Arc constraint(s)	<ul style="list-style-type: none"> • <code>in_set(vars.idvar, vals.val) \Leftrightarrow in_set(vals.idval, vars.var)</code> • <code>vars.nocc = card_set(vars.var)</code> • <code>vals.nocc = card_set(vals.val)</code>
Graph property(ies)	<code>NARC = VARS * VALS </code>

Example	$\text{symmetric_gcc} \left(\left(\left\{ \begin{array}{l} \text{idvar} - 1 \quad \text{var} - \{3\} \quad \text{nocc} - 1, \\ \text{idvar} - 2 \quad \text{var} - \{1\} \quad \text{nocc} - 1, \\ \text{idvar} - 3 \quad \text{var} - \{1, 2\} \quad \text{nocc} - 2, \\ \text{idvar} - 4 \quad \text{var} - \{1, 3\} \quad \text{nocc} - 2 \end{array} \right\}, \right. \right. \\ \left. \left. \left\{ \begin{array}{l} \text{idval} - 1 \quad \text{val} - \{2, 3, 4\} \quad \text{nocc} - 3, \\ \text{idval} - 2 \quad \text{val} - \{3\} \quad \text{nocc} - 1, \\ \text{idval} - 3 \quad \text{val} - \{1, 4\} \quad \text{nocc} - 2, \\ \text{idval} - 4 \quad \text{val} - \emptyset \quad \text{nocc} - 0 \end{array} \right\} \right) \right)$
----------------	--

Parts (A) and (B) of Figure 4.411 respectively show the initial and final graph. Since we use the **NARC** graph property, all the arcs of the final graph are stressed in bold.

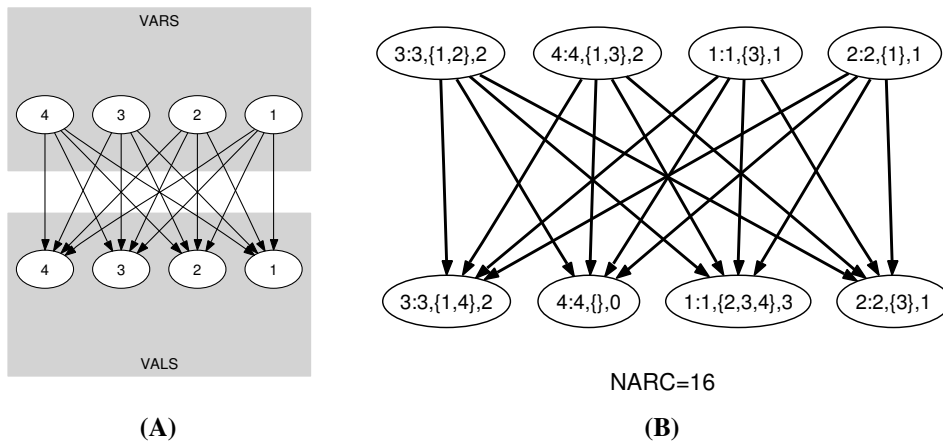


Figure 4.411: Initial and final graph of the `symmetric_gcc` constraint

Graph model

The graph model used for the `symmetric_gcc` is similar to the one used in the `domain_constraint` or in the `link_set_to_booleans` constraints: We use an equivalence in the arc constraint and ask all arc constraints to hold.

Signature

Since we use the *PRODUCT* arc generator on the collections `VARs` and `VALs`, the number of arcs of the initial graph is equal to $|\text{VARs}| \cdot |\text{VALs}|$. Therefore the maximum number of arcs of the final graph is also equal to $|\text{VARs}| \cdot |\text{VALs}|$ and we can rewrite **NARC** = $|\text{VARs}| \cdot |\text{VALs}|$ to **NARC** $\geq |\text{VARs}| \cdot |\text{VALs}|$. So we can simplify **NARC** to **NARC**.

Usage

The most simple example of applying `symmetric_gcc` is a variant of personnel assignment problem, where one person can be assigned to perform between n and m ($n \leq m$) jobs, and every job requires between p and q ($p \leq q$) persons. In addition every job requires different kind of skills. The previous problem can be modelled as follows:

- For each person we create an item of the `VARs` collection,
- For each job we create an item of the `VALs` collection,
- There is an arc between a person and the particular job if this person is qualified to perform it.

Remark

The `symmetric_gcc` constraint generalizes the `global_cardinality` constraint by allowing a variable to take more than one value. It corresponds to a variant of the `symmetric_cardinality` constraint described in [154] where the occurrence variables of the `VARs` and `VALs` collections are replaced by fixed intervals.

See also

`symmetric_cardinality`, `global_cardinality`, `link_set_to_booleans`.

Key words

decomposition, timetabling constraint, assignment, relation, flow, constraint involving set variables.

4.218 temporal_path

Origin	ILOG
Constraint	temporal_path(NPATH, NODES)
Argument(s)	NPATH : dvar NODES : collection(index - int, succ - dvar, start - dvar, end - dvar)
Restriction(s)	NPATH ≥ 1 NPATH $\leq \text{NODES} $ required(NODES, [index, succ, start, end]) $ \text{NODES} > 0$ NODES.index ≥ 1 NODES.index $\leq \text{NODES} $ distinct(NODES, index) NODES.succ ≥ 1 NODES.succ $\leq \text{NODES} $

Purpose

Let G be the digraph described by the NODES collection. Partition G with a set of disjoint paths such that each vertex of the graph belongs to a single path. In addition, for all pairs of consecutive vertices of a path we have a precedence constraint that enforces the end associated to the first vertex to be less than or equal to the start related to the second vertex.

Arc input(s)	NODES
Arc generator	<i>CLIQUE</i> \mapsto collection(nodes1, nodes2)
Arc arity	2
Arc constraint(s)	<ul style="list-style-type: none"> • nodes1.succ = nodes2.index • nodes1.succ = nodes1.index \vee nodes1.end \leq nodes2.start • nodes1.start \leq nodes1.end • nodes2.start \leq nodes2.end
Graph property(ies)	<ul style="list-style-type: none"> • MAX_ID = 1 • NCC = NPATH • NVERTEX = NODES

Example

$$\text{temporal_path} \left(2, \left\{ \begin{array}{l} \left(\begin{array}{cccc} \text{index} - 1 & \text{succ} - 2 & \text{start} - 0 & \text{end} - 1, \\ \text{index} - 2 & \text{succ} - 6 & \text{start} - 3 & \text{end} - 5, \\ \text{index} - 3 & \text{succ} - 4 & \text{start} - 0 & \text{end} - 3, \\ \text{index} - 4 & \text{succ} - 5 & \text{start} - 4 & \text{end} - 6, \\ \text{index} - 5 & \text{succ} - 7 & \text{start} - 7 & \text{end} - 8, \\ \text{index} - 6 & \text{succ} - 6 & \text{start} - 7 & \text{end} - 9, \\ \text{index} - 7 & \text{succ} - 7 & \text{start} - 9 & \text{end} - 10 \end{array} \right) \end{array} \right. \right\} \right)$$

Parts (A) and (B) of Figure 4.412 respectively show the initial and final graph. Since we use the MAX_ID, the NCC and the NVERTEX graph properties we display the following information on the final graph:

- We show with a double circle a vertex which has the maximum number of predecessors.
- We show the two connected components corresponding to the two paths.
- We put in bold the vertices.

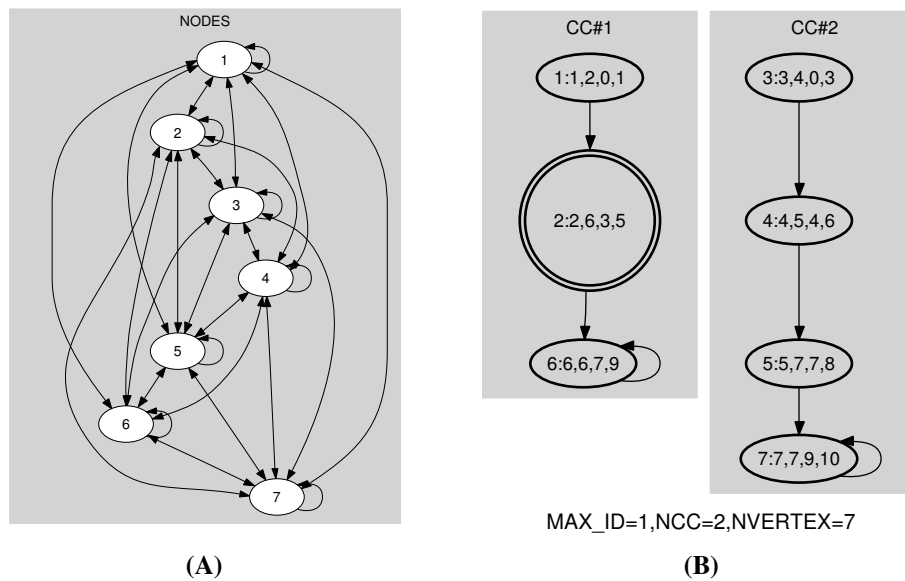


Figure 4.412: Initial and final graph of the `temporal_path` constraint

Graph model

The arc constraint is a conjunction of four conditions that respectively correspond to:

- A constraint that links the successor variable of a first vertex to the index attribute of a second vertex,
- A precedence constraint that applies on one vertex and its distinct successor,
- One precedence constraint between the start and the end of the vertex that corresponds to the departure of an arc,
- One precedence constraint between the start and the end of the vertex that corresponds to the arrival of an arc.

We use the following three graph properties in order to enforce the partitioning of the graph in distinct paths:

- The first property $\text{MAX_ID} = 1$ enforces that each vertex has only one single predecessor (except the last vertex of a path which has also itself as a predecessor),
- The second property $\text{NCC} = \text{NPATH}$ ensures that we have the required number of paths,
- The third property $\text{NVERTEX} = |\text{NODES}|$ enforces that for each vertex, the start is not located after the end.

- Signature** Since we use the graph property $NVERTEX = |NODES|$ together with the restriction $|NODES| > 0$ the final graph is not empty. Therefore the smallest possible value of MAX_ID is equal to 1. So we can rewrite $MAX_ID = 1$ to $MAX_ID \leq 1$ and simplify MAX_ID to MAX_ID.
- Since the maximum number of vertices of the final graph is equal to $|NODES|$ we can rewrite the graph property $NVERTEX = |NODES|$ to $NVERTEX \geq |NODES|$ and simplify NVERTEX to NVERTEX.
- Remark** This constraint is related to the `path` constraint of Ilog Solver. It can also be directly expressed with the `cycle` [37] constraint of CHIP by using the `diff` nodes and the origin parameters. A generic model based on linear programming that handles paths, trees and cycles is presented in [94].
- See also** `path_from_to`.
- Key words** graph constraint, graph partitioning constraint, path, connected component.

20000128

895

4.219 tour

Origin	[74]
Constraint	tour(NODES)
Synonym(s)	atour, cycle.
Argument(s)	NODES : collection(index – int, succ – svar)
Restriction(s)	$ \text{NODES} \geq 3$ required(NODES, [index, succ]) $\text{NODES.index} \geq 1$ $\text{NODES.index} \leq \text{NODES} $ distinct(NODES, index)
Purpose	Enforce to cover an undirected graph G described by the NODES collection with a Hamiltonian cycle.
Arc input(s)	NODES
Arc generator	$\overline{\text{CLIQUE}(\neq)} \mapsto \text{collection}(\text{nodes1}, \text{nodes2})$
Arc arity	2
Arc constraint(s)	$\text{in_set}(\text{nodes2.index}, \text{nodes1.succ}) \Leftrightarrow \text{in_set}(\text{nodes1.index}, \text{nodes2.succ})$
Graph property(ies)	$\overline{\text{NARC}} = \text{NODES} * \text{NODES} - \text{NODES} $
Arc input(s)	NODES
Arc generator	$\overline{\text{CLIQUE}(\neq)} \mapsto \text{collection}(\text{nodes1}, \text{nodes2})$
Arc arity	2
Arc constraint(s)	$\text{in_set}(\text{nodes2.index}, \text{nodes1.succ})$
Graph property(ies)	<ul style="list-style-type: none"> • $\overline{\text{MIN_NSCC}} = \text{NODES}$ • $\overline{\text{MIN_ID}} = 2$ • $\overline{\text{MAX_ID}} = 2$ • $\overline{\text{MIN_OD}} = 2$ • $\overline{\text{MAX_OD}} = 2$

Example	$\text{tour} \left(\left(\begin{array}{l} \left\{ \begin{array}{ll} \text{index} - 1 & \text{succ} - \{2, 4\}, \\ \text{index} - 2 & \text{succ} - \{1, 3\}, \\ \text{index} - 3 & \text{succ} - \{2, 4\}, \\ \text{index} - 4 & \text{succ} - \{1, 3\} \end{array} \right\} \end{array} \right) \right)$
----------------	---

Part (A) of Figure 4.413 shows the initial graph from which we start. It is derived from the set associated to each vertex. Each set describes the potential values of the succ attribute of a given vertex. Part (B) of Figure 4.413 gives the final graph associated to the example. The tour constraint holds since the final graph corresponds to a Hamiltonian cycle.

Graph model

The first graph property enforces the subsequent condition: If we have an arc from the i^{th} vertex to the j^{th} vertex then we have also an arc from the j^{th} vertex to the i^{th} vertex. The second graph property enforces the following constraints:

- We have one strongly connected component containing $|\text{NODES}|$ vertices,
- Each vertex has exactly two predecessors and two successors.

Signature

Since the maximum number of vertices of the final graph is equal to $|\text{NODES}|$, we can rewrite the graph property $\text{MIN_NSCC} = |\text{NODES}|$ to $\text{MIN_NSCC} \geq |\text{NODES}|$ and simplify MIN_NSCC to MIN_NSCC.

See also

circuit, cycle, link_set_to_booleans.

Key words

graph constraint, undirected graph, Hamiltonian, linear programming, constraint involving set variables.

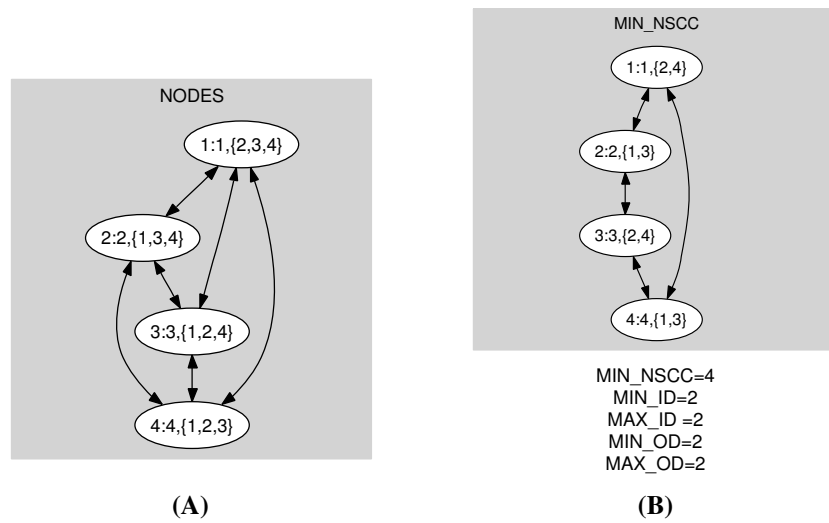


Figure 4.413: Initial and final graph of the tour set constraint

20030820

899

4.220 track

Origin	[155]
Constraint	<code>track(NTRAIL, TASKS)</code>
Argument(s)	<code>NTRAIL : int</code> <code>TASKS : collection(trail - int, origin - dvar, end - dvar)</code>
Restriction(s)	<code>NTRAIL > 0</code> <code>required(TASKS, [trail, origin, end])</code> <code>TASKS.trail > 0</code> <code>TASKS.trail ≤ NTRAIL</code>
Purpose	The track constraint enforces that, at each point in time overlapped by at least one task, the number of distinct values of the trail attribute of the set of tasks that overlap that point, is equal to NTRAIL.
Derived Collection(s)	$\text{col} \left(\begin{array}{l} \text{TIME_POINTS} - \text{collection}(\text{origin} - \text{dvar}, \text{end} - \text{dvar}, \text{point} - \text{dvar}), \\ \left[\begin{array}{l} \text{item}(\text{origin} - \text{TASKS.origin}, \text{end} - \text{TASKS.end}, \text{point} - \text{TASKS.origin}), \\ \text{item}(\text{origin} - \text{TASKS.origin}, \text{end} - \text{TASKS.end}, \text{point} - \text{TASKS.end} - 1) \end{array} \right] \end{array} \right)$
Arc input(s)	<code>TASKS</code>
Arc generator	<code>SELF ↦ collection(tasks)</code>
Arc arity	1
Arc constraint(s)	<code>tasks.origin ≤ tasks.end</code>
Graph property(ies)	$\overline{\text{NARC}} = \text{TASKS} $
Arc input(s)	<code>TIME_POINTS TASKS</code>
Arc generator	<code>PRODUCT ↦ collection(time_points, tasks)</code>
Arc arity	2
Arc constraint(s)	<ul style="list-style-type: none"> • <code>time_points.end > time_points.origin</code> • <code>tasks.origin ≤ time_points.point</code> • <code>time_points.point < tasks.end</code>
Sets	<code>SUCC ↦</code> $\left[\begin{array}{l} \text{source}, \\ \text{variables} - \text{col}(\text{VARIABLES} - \text{collection}(\text{var} - \text{dvar}), [\text{item}(\text{var} - \text{TASKS.trail})]) \end{array} \right]$
Constraint(s) on sets	<code>nvalue(NTRAIL, variables)</code>

Example

$$\text{track} \left(2, \left(\begin{array}{l} \text{trail} - 1 \quad \text{origin} - 1 \quad \text{end} - 2, \\ \text{trail} - 2 \quad \text{origin} - 1 \quad \text{end} - 2, \\ \text{trail} - 1 \quad \text{origin} - 2 \quad \text{end} - 4, \\ \text{trail} - 2 \quad \text{origin} - 2 \quad \text{end} - 3, \\ \text{trail} - 2 \quad \text{origin} - 3 \quad \text{end} - 4 \end{array} \right) \right)$$

The previous constraint holds since:

- The first and second tasks both overlap instant 1 and have a respective trail of 1 and 2, which makes two distinct values for the trail attribute at instant 1,
- The third and fourth tasks both overlap instant 2 and have a respective trail of 1 and 2, which makes two distinct values for the trail attribute at instant 2,
- The third and fifth tasks both overlap instant 3 and have a respective trail of 1 and 2, which makes two distinct values for the trail attribute at instant 3.

Parts (A) and (B) of Figure 4.414 respectively show the initial and final graph of the second graph constraint.

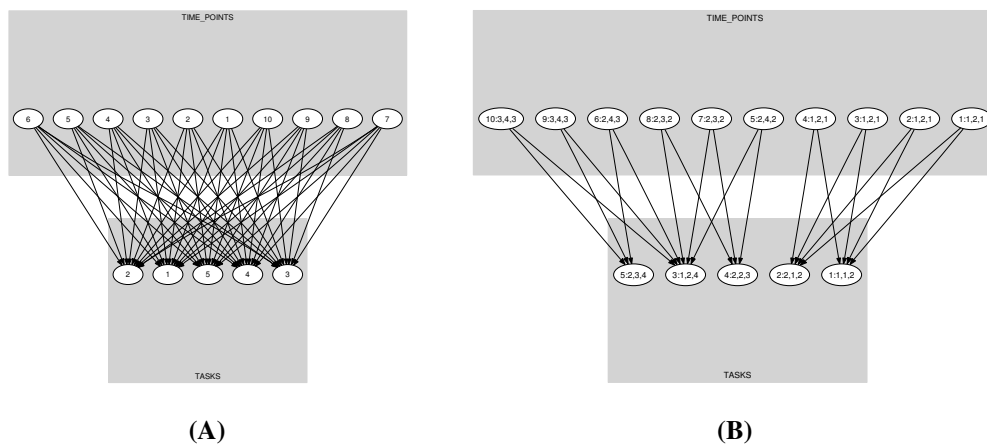


Figure 4.414: Initial and final graph of the `track` constraint

Signature

Consider the first graph constraint. Since we use the *SELF* arc generator on the `TASKS` collection, the maximum number of arcs of the final graph is equal to $|\text{TASKS}|$. Therefore we can rewrite $\text{NARC} = |\text{TASKS}|$ to $\text{NARC} \geq |\text{TASKS}|$ and simplify NARC to NARC.

See also

`nvalue`.

Key words

timetabling constraint, resource constraint, temporal constraint, derived collection.

4.221 tree

Origin	N. Beldiceanu
Constraint	tree(NTREES, NODES)
Argument(s)	NTREES : dvar NODES : collection(index - int, succ - dvar)
Restriction(s)	NTREES \geq 0 required(NODES, [index, succ]) NODES.index \geq 1 NODES.index \leq NODES distinct(NODES, index) NODES.succ \geq 1 NODES.succ \leq NODES
Purpose	Cover a digraph G by a set of trees in such a way that each vertex of G belongs to one distinct tree. The edges of the trees are directed from their leaves to their respective roots.
Arc input(s)	NODES
Arc generator	<i>CLIQUE</i> \mapsto collection(nodes1, nodes2)
Arc arity	2
Arc constraint(s)	nodes1.succ = nodes2.index
Graph property(ies)	<ul style="list-style-type: none"> • $\text{MAX_NSCC} \leq 1$ • $\text{NCC} = \text{NTREES}$

Example	$\text{tree} \left(2, \left\{ \begin{array}{l} \text{index} - 1 \quad \text{succ} - 1, \\ \text{index} - 2 \quad \text{succ} - 5, \\ \text{index} - 3 \quad \text{succ} - 5, \\ \text{index} - 4 \quad \text{succ} - 7, \\ \text{index} - 5 \quad \text{succ} - 1, \\ \text{index} - 6 \quad \text{succ} - 1, \\ \text{index} - 7 \quad \text{succ} - 7, \\ \text{index} - 8 \quad \text{succ} - 5 \end{array} \right\} \right)$
----------------	---

Parts (A) and (B) of Figure 4.415 respectively show the initial and final graph. Since we use the **NCC** graph property, we display the two connected components of the final graph. Each of them corresponds to a tree. The **tree** constraint holds since all strongly connected components of the final graph have no more than one vertex and since $\text{NTREES} = \text{NCC} = 2$.

Graph model	We use the graph property $\text{MAX_NSCC} \leq 1$ in order to specify the fact that the size of the largest strongly connected component should not exceed one. In fact each root of a tree is a strongly connected component with one single vertex. The second graph property $\text{NCC} = \text{NTREES}$ enforces the number of trees to be equal to the number of connected components.
--------------------	--

Algorithm

An arc-consistency filtering algorithm for the `tree` constraint is described in [156]. This algorithm is based on a necessary and sufficient condition that we now depict.

To any `tree` constraint we associate the digraph $G = (V, E)$, where:

- To each item `NODES[i]` of the `NODES` collection corresponds a vertex v_i of G .
- For every pair of items (`NODES[i]`, `NODES[j]`) of the `NODES` collection, where i and j are not necessarily distinct, there is an arc from v_i to v_j in E if j is a potential value of `NODES[i].succ`.

A strongly connected component C of G is called a *sink component* if all the successors of all vertices of C belong to C . Let `MINTREES` and `MAXTREES` respectively denote the number of sink components of G and the number of vertices of G with a loop.

The `tree` constraint has a solution if and only if:

- Each sink component of G contains at least one vertex with a loop,
- The domain of `NTREES` has at least one value within interval `[MINTREES, MAXTREES]`.

See also

`binary_tree`, `cycle`, `map`, `tree_resource`, `graph_crossing`.

Key words

graph constraint, graph partitioning constraint, connected component, tree, `one_succ`.

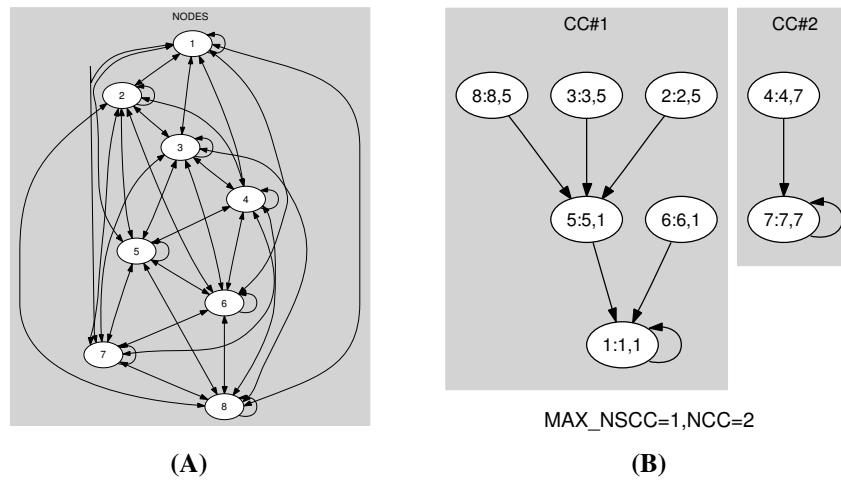


Figure 4.415: Initial and final graph of the tree constraint

20000128

905

4.222 tree_range

Origin	Derived from <i>tree</i> .
Constraint	<code>tree_range(NTREES, R, NODES)</code>
Argument(s)	<p>NTREES : dvar R : dvar NODES : collection(index – int, succ – dvar)</p>
Restriction(s)	<p>NTREES ≥ 0 R ≥ 0 R $< \text{NODES}$ required(NODES, [index, succ]) NODES.index ≥ 1 NODES.index $\leq \text{NODES}$ distinct(NODES, index) NODES.succ ≥ 1 NODES.succ $\leq \text{NODES}$</p>
Purpose	<p>Cover the digraph G described by the NODES collection with NTREES trees in such a way that each vertex of G belongs to one distinct tree. R is the difference between the longest and the shortest paths of the final graph.</p>
Arc input(s)	NODES
Arc generator	<i>CLIQUE</i> \mapsto collection(nodes1, nodes2)
Arc arity	2
Arc constraint(s)	nodes1.succ = nodes2.index
Graph property(ies)	<ul style="list-style-type: none"> • MAX_NSCC ≤ 1 • NCC = NTREES • RANGE_DRG = R

Example `tree_range` $\left(2, 1, \left\{ \begin{array}{l} \text{index} - 1 \quad \text{succ} - 1, \\ \text{index} - 2 \quad \text{succ} - 5, \\ \text{index} - 3 \quad \text{succ} - 5, \\ \text{index} - 4 \quad \text{succ} - 7, \\ \text{index} - 5 \quad \text{succ} - 1, \\ \text{index} - 6 \quad \text{succ} - 1, \\ \text{index} - 7 \quad \text{succ} - 7, \\ \text{index} - 8 \quad \text{succ} - 5 \end{array} \right\} \right)$

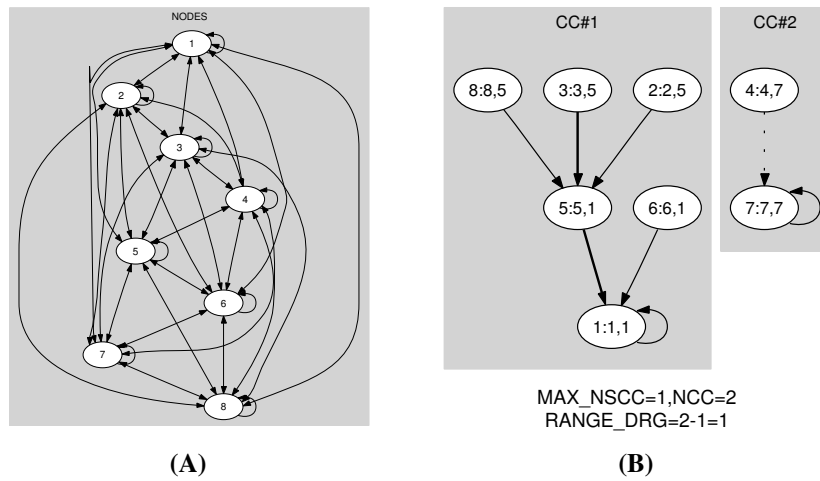
Parts (A) and (B) of Figure 4.416 respectively show the initial and final graph. Since we use the RANGE_DRG graph property, we respectively display the longest and shortest paths of the final graph with a bold and a dash line.

20030820

907

See also tree, balance.

Key words graph constraint, graph partitioning constraint, connected component, tree, balanced tree.

Figure 4.416: Initial and final graph of the `tree_range` constraint

20030820

909

4.223 tree_resource

Origin	Derived from tree.
Constraint	tree_resource(RESOURCE, TASK)
Argument(s)	RESOURCE : collection(id – int, nb_task – dvar) TASK : collection(id – int, father – dvar, resource – dvar)
Restriction(s)	required(RESOURCE, [id, nb_task]) RESOURCE.id ≥ 1 RESOURCE.id $\leq \text{RESOURCE} $ distinct(RESOURCE, id) RESOURCE.nb_task ≥ 0 RESOURCE.nb_task $\leq \text{TASK} $ required(TASK, [id, father, resource]) TASK.id $> \text{RESOURCE} $ TASK.id $\leq \text{RESOURCE} + \text{TASK} $ distinct(TASK, id) TASK.father ≥ 1 TASK.father $\leq \text{RESOURCE} + \text{TASK} $ TASK.resource ≥ 1 TASK.resource $\leq \text{RESOURCE} $
Purpose	Cover a digraph G in such a way that each vertex belongs to one distinct tree. Each tree is made up from one <i>resource</i> vertex and several <i>task</i> vertices. The resource vertices correspond to the roots of the different trees. For each resource a domain variable nb_task indicates how many task-vertices belong to the corresponding tree. For each task a domain variable resource gives the identifier of the resource which can handle that task.
Derived Collection(s)	$\text{col} \left(\begin{array}{l} \text{RESOURCE_TASK} - \text{collection}(\text{index} - \text{int}, \text{succ} - \text{dvar}, \text{name} - \text{dvar}), \\ \left[\begin{array}{l} \text{item}(\text{index} - \text{RESOURCE.id}, \text{succ} - \text{RESOURCE.id}, \text{name} - \text{RESOURCE.id}), \\ \text{item}(\text{index} - \text{TASK.id}, \text{succ} - \text{TASK.father}, \text{name} - \text{TASK.resource}) \end{array} \right] \end{array} \right)$
Arc input(s)	RESOURCE_TASK
Arc generator	CLIQUE \mapsto collection(resource_task1, resource_task2)
Arc arity	2
Arc constraint(s)	<ul style="list-style-type: none"> • resource_task1.succ = resource_task2.index • resource_task1.name = resource_task2.name
Graph property(ies)	<ul style="list-style-type: none"> • MAX_NSICC ≤ 1 • NCC = RESOURCE • NVERTEX = RESOURCE + TASK
For all items of RESOURCE:	

Arc input(s)	RESOURCE_TASK
Arc generator	$CLIQUE \mapsto \text{collection}(\text{resource_task1}, \text{resource_task2})$
Arc arity	2
Arc constraint(s)	<ul style="list-style-type: none"> • $\text{resource_task1.succ} = \text{resource_task2.index}$ • $\text{resource_task1.name} = \text{resource_task2.name}$ • $\text{resource_task1.name} = \text{RESOURCE.id}$
Graph property(ies)	$NVERTEX = \text{RESOURCE.nb_task} + 1$

Example	$\text{tree_resource} \left(\begin{array}{l} \left\{ \begin{array}{l} \text{id} - 1 \quad \text{nb_task} - 4, \\ \text{id} - 2 \quad \text{nb_task} - 0, \\ \text{id} - 3 \quad \text{nb_task} - 1 \end{array} \right\}, \\ \left\{ \begin{array}{l} \text{id} - 4 \quad \text{father} - 8 \quad \text{resource} - 1, \\ \text{id} - 5 \quad \text{father} - 3 \quad \text{resource} - 3, \\ \text{id} - 6 \quad \text{father} - 8 \quad \text{resource} - 1, \\ \text{id} - 7 \quad \text{father} - 1 \quad \text{resource} - 1, \\ \text{id} - 8 \quad \text{father} - 1 \quad \text{resource} - 1 \end{array} \right\} \end{array} \right)$
----------------	---

For the second graph constraint, part (A) of Figure 4.417 shows the initial graphs associated to resources 1, 2 and 3. For the second graph constraint, part (B) of Figure 4.417 shows the final graphs associated to resources 1, 2 and 3. Since we use the **NVERTEX** graph property, the vertices of the final graphs are stressed in bold. To each resource corresponds a tree of respectively 4, 0 and 1 task-vertices.

Signature	Since the initial graph of the first graph constraint contains $ \text{RESOURCE} + \text{TASK} $ vertices, the corresponding final graph cannot have more than $ \text{RESOURCE} + \text{TASK} $ vertices. Therefore we can rewrite the graph property $\mathbf{NVERTEX} = \text{RESOURCE} + \text{TASK} $ to $\mathbf{NVERTEX} \geq \text{RESOURCE} + \text{TASK} $ and simplify <u>$\mathbf{NVERTEX}$</u> to $\mathbf{NVERTEX}$.
------------------	--

See also	tree.
-----------------	-------

Key words	graph constraint, tree, resource constraint, graph partitioning constraint, connected component, derived collection.
------------------	--

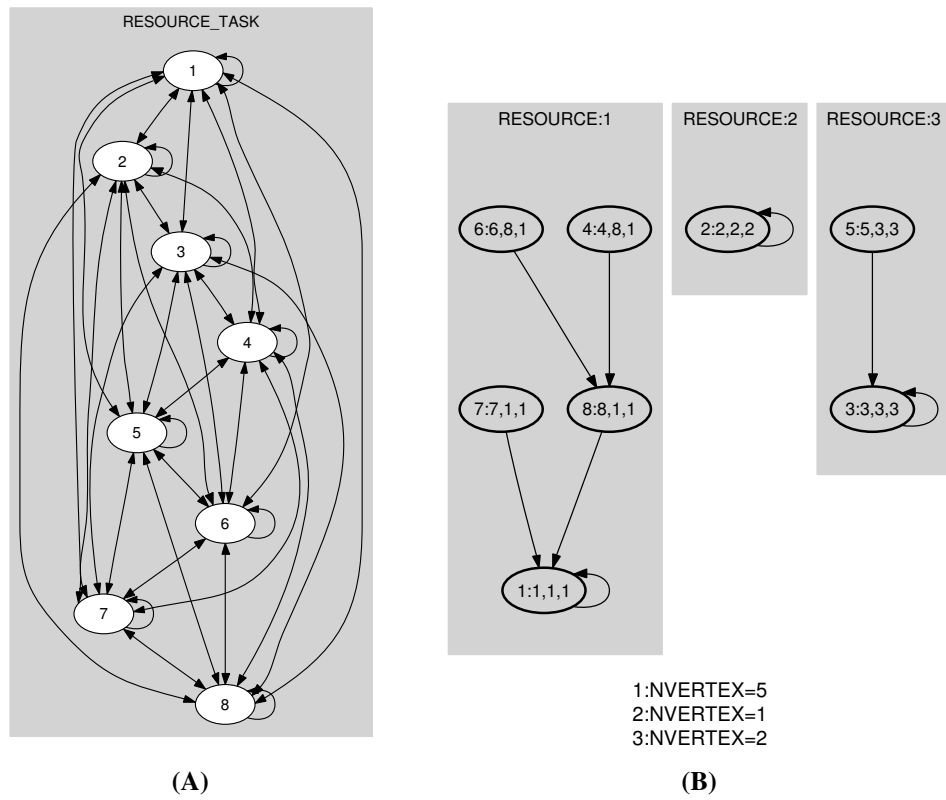


Figure 4.417: Initial and final graph of the tree_resource constraint

20030820

913

4.224 two_layer_edge_crossing

Origin	Inspired by [157].
Constraint	<code>two_layer_edge_crossing(NCROSS, VERTICES_LAYER1, VERTICES_LAYER2, EDGES)</code>
Argument(s)	<pre> NCROSS : dvar VERTICES_LAYER1 : collection(id - int, pos - dvar) VERTICES_LAYER2 : collection(id - int, pos - dvar) EDGES : collection(id - int, vertex1 - int, vertex2 - int) </pre>
Restriction(s)	<pre> NCROSS ≥ 0 required(VERTICES_LAYER1, [id, pos]) VERTICES_LAYER1.id ≥ 1 VERTICES_LAYER1.id ≤ VERTICES_LAYER1 distinct(VERTICES_LAYER1, id) required(VERTICES_LAYER2, [id, pos]) VERTICES_LAYER2.id ≥ 1 VERTICES_LAYER2.id ≤ VERTICES_LAYER2 distinct(VERTICES_LAYER2, id) required(EDGES, [id, vertex1, vertex2]) EDGES.id ≥ 1 EDGES.id ≤ EDGES distinct(EDGES, id) EDGES.vertex1 ≥ 1 EDGES.vertex1 ≤ VERTICES_LAYER1 EDGES.vertex2 ≥ 1 EDGES.vertex2 ≤ VERTICES_LAYER2 </pre>
Purpose	NCROSS is the number of line-segments intersections.
Derived Collection(s)	$\text{col} \left(\begin{array}{c} \text{EDGES_EXTREMITIES} - \text{collection}(\text{layer1} - \text{dvar}, \text{layer2} - \text{dvar}), \\ \left[\text{item} \left(\begin{array}{c} \text{layer1} - \text{EDGES.vertex1}(\text{VERTICES_LAYER1}, \text{pos}, \text{id}), \\ \text{layer2} - \text{EDGES.vertex2}(\text{VERTICES_LAYER2}, \text{pos}, \text{id}) \end{array} \right) \right] \end{array} \right)$
Arc input(s)	EDGES_EXTREMITIES
Arc generator	$CLIQUE(\langle) \mapsto \text{collection}(\text{edges_extremities1}, \text{edges_extremities2})$
Arc arity	2
Arc constraint(s)	$\bigvee \left(\begin{array}{c} \bigwedge \left(\begin{array}{c} \text{edges_extremities1.layer1} < \text{edges_extremities2.layer1}, \\ \text{edges_extremities1.layer2} > \text{edges_extremities2.layer2} \end{array} \right), \\ \bigwedge \left(\begin{array}{c} \text{edges_extremities1.layer1} > \text{edges_extremities2.layer1}, \\ \text{edges_extremities1.layer2} < \text{edges_extremities2.layer2} \end{array} \right) \end{array} \right)$
Graph property(ies)	$NARC = NCROSS$

Example

$$\text{two_layer_edge_crossing} \left(\begin{array}{l} 2, \{ \text{id} - 1 \text{ pos} - 1, \text{id} - 2 \text{ pos} - 2 \}, \\ \left\{ \begin{array}{l} \text{id} - 1 \text{ pos} - 3, \\ \text{id} - 2 \text{ pos} - 1, \\ \text{id} - 3 \text{ pos} - 2 \end{array} \right\}, \\ \left\{ \begin{array}{l} \text{id} - 1 \text{ vertex1} - 2 \text{ vertex2} - 2, \\ \text{id} - 2 \text{ vertex1} - 2 \text{ vertex2} - 3, \\ \text{id} - 3 \text{ vertex1} - 1 \text{ vertex2} - 1 \end{array} \right\} \end{array} \right)$$

Parts (A) and (B) of Figure 4.418 respectively show the initial and final graph. Since we use the **NARC** graph property, the arcs of the final graph are stressed in bold. Figure 4.419 gives a picture of the previous example, where one can observe the two line-segments intersections. Each line-segment of Figure 4.419 is labelled with its identifier and corresponds to one vertex of the initial and final graph depicted in Figure 4.418.

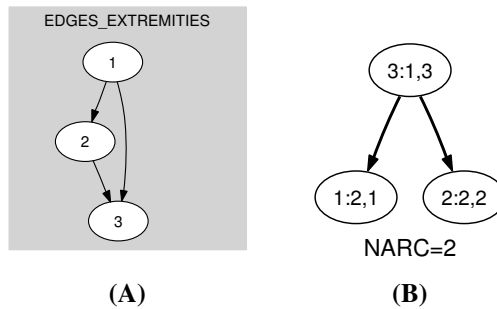


Figure 4.418: Initial and final graph of the two_layer_edge_crossing constraint

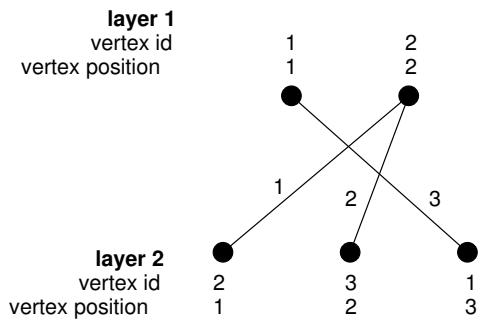


Figure 4.419: Intersection between line-segments joining two layers

Graph model

As usual for the two-layer edge crossing problem [157], [158], positions of the vertices on each layer are represented as a permutation of the vertices. We generate a derived collection which, for each edges, contains the position of its extremities on both layers. In the arc generator we use the restriction $<$ in order to generate one single arc for each pair of segments. This is required, since otherwise we would count more than once a line-segments intersection.

- Remark** The two-layer edge crossing minimization problem was proved to be NP-hard in [159].
- See also** `crossing`, `graph_crossing`.
- Key words** geometrical constraint, line-segments intersection, derived collection.

20030820

917

4.225 two_orth_are_in_contact

Origin	Used for defining <code>orths_are_connected</code> .
Constraint	<code>two_orth_are_in_contact(ORTHOTOPE1, ORTHOTOPE2)</code>
Type(s)	<code>ORTHOTOPE : collection(ori - dvar, siz - dvar, end - dvar)</code>
Argument(s)	<code>ORTHOTOPE1 : ORTHOTOPE</code> <code>ORTHOTOPE2 : ORTHOTOPE</code>
Restriction(s)	<code> ORTHOTOPE > 0</code> <code>require_at_least(2, ORTHOTOPE, [ori, siz, end])</code> <code>ORTHOTOPE.siz > 0</code> <code> ORTHOTOPE1 = ORTHOTOPE2 </code> <code>orth_link_ori_siz_end(ORTHOTOPE1)</code> <code>orth_link_ori_siz_end(ORTHOTOPE2)</code>
Purpose	<p>Enforce the following conditions on two orthotopes O_1 and O_2:</p> <ul style="list-style-type: none"> • For all dimensions i, except one dimension, the projections of O_1 and O_2 on i have a non-empty intersection. • For all dimensions i, the distance between the projections of O_1 and O_2 on i is equal to 0.
Arc input(s)	<code>ORTHOTOPE1 ORTHOTOPE2</code>
Arc generator	<code>PRODUCT(=) ↦ collection(orthotope1, orthotope2)</code>
Arc arity	2
Arc constraint(s)	<ul style="list-style-type: none"> • <code>orthotope1.end > orthotope2.ori</code> • <code>orthotope2.end > orthotope1.ori</code>
Graph property(ies)	<code>NARC = ORTHOTOPE1 - 1</code>
Arc input(s)	<code>ORTHOTOPE1 ORTHOTOPE2</code>
Arc generator	<code>PRODUCT(=) ↦ collection(orthotope1, orthotope2)</code>
Arc arity	2
Arc constraint(s)	<code>max (0, max(orthotope1.ori, orthotope2.ori) - min(orthotope1.end, orthotope2.end)) = 0</code>
Graph property(ies)	<code>NARC = ORTHOTOPE1 </code>

Example

$$\text{two_orth_are_in_contact} \left(\left(\left\{ \begin{array}{l} \text{ori} - 1 \quad \text{siz} - 3 \quad \text{end} - 4, \\ \text{ori} - 5 \quad \text{siz} - 2 \quad \text{end} - 7, \\ \text{ori} - 3 \quad \text{siz} - 2 \quad \text{end} - 5, \\ \text{ori} - 2 \quad \text{siz} - 3 \quad \text{end} - 5 \end{array} \right\} \right) \right)$$

Parts (A) and (B) of Figure 4.420 respectively show the initial and final graph associated to the first graph constraint. Since we use the **NARC** graph property, the unique arc of the final graph is stressed in bold. It corresponds to the fact that the projection in dimension 1 of the two rectangles of the example overlap. Figure 4.421 shows the two rectangles of the previous example.

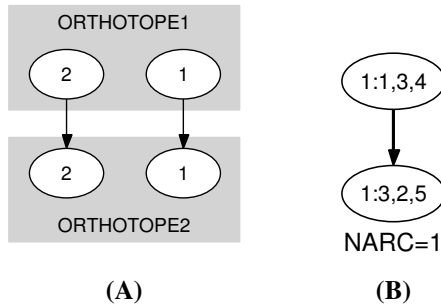


Figure 4.420: Initial and final graph of the `two_orth_are_in_contact` constraint

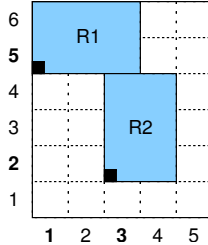


Figure 4.421: Two connected rectangles

Signature

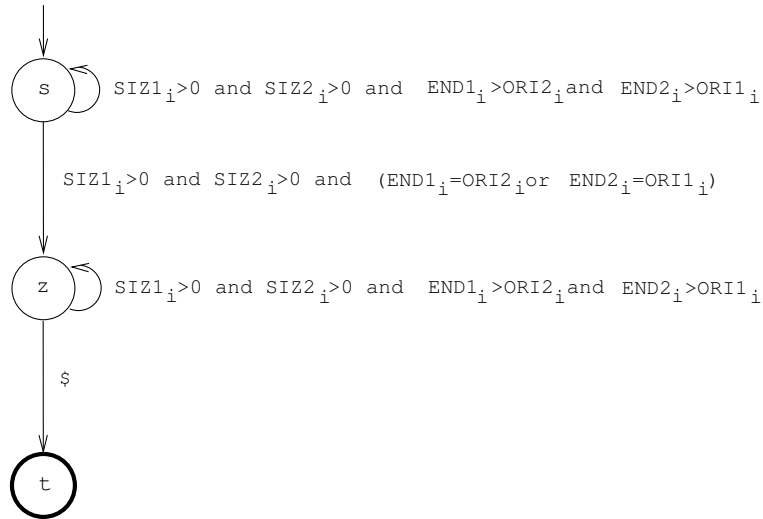
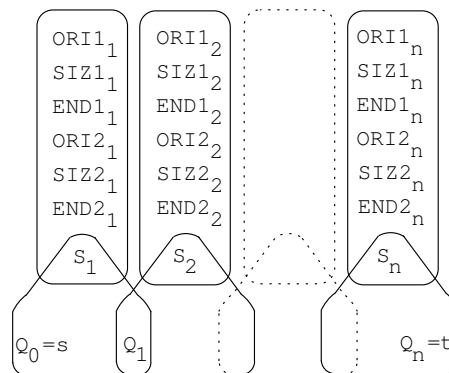
Consider the second graph constraint. Since we use the arc generator *PRODUCT*(=) on the collections ORTHOTOPE1 and ORTHOTOPE2, and because of the restriction $|\text{ORTHOTOPE1}| = |\text{ORTHOTOPE2}|$, the maximum number of arcs of the corresponding final graph is equal to $|\text{ORTHOTOPE1}|$. Therefore we can rewrite the graph property **NARC** = $|\text{ORTHOTOPE1}|$ to **NARC** $\geq |\text{ORTHOTOPE1}|$ and simplify NARC to **NARC**.

Automaton

Figure 4.422 depicts the automaton associated to the `two_orth_are_in_contact` constraint. Let ORI1_i , SIZ1_i and END1_i respectively be the *ori*, the *siz* and the *end* attributes of the i^{th} item of the ORTHOTOPE1 collection. Let ORI2_i , SIZ2_i and END2_i respectively be the *ori*, the *siz* and the *end* attributes of the i^{th} item of the ORTHOTOPE2 collection. To each sextuple $(\text{ORI1}_i, \text{SIZ1}_i, \text{END1}_i, \text{ORI2}_i, \text{SIZ2}_i, \text{END2}_i)$ corresponds a signature variable S_i , which takes its value in $\{0, 1, 2\}$, as well as the following signature constraint:

$$((\text{SIZ1}_i > 0) \wedge (\text{SIZ2}_i > 0) \wedge (\text{END1}_i > \text{ORI2}_i) \wedge (\text{END2}_i > \text{ORI1}_i)) \Leftrightarrow S_i = 0$$

$$((SIZ1_i > 0) \wedge (SIZ2_i > 0) \wedge (END1_i = ORI2_i \vee END2_i = ORI1_i)) \Leftrightarrow S_i = 1.$$

Figure 4.422: Automaton of the `two_orth_are_in_contact` constraintFigure 4.423: Hypergraph of the reformulation corresponding to the automaton of the `two_orth_are_in_contact` constraint


Used in `orths_are_connected`.

Key words geometrical constraint, touch, contact, non-overlapping, orthotope, Berge-acyclic constraint network, automaton, automaton without counters.

20030820

921

4.226 two_orth_column

Origin	Used for defining <code>diffn_column</code> .
Constraint	<code>two_orth_column(ORTHOPE1, ORTHOTOPE2, N)</code>
Type(s)	<code>ORTHOPE : collection(ori - dvar, siz - dvar, end - dvar)</code>
Argument(s)	<code>ORTHOPE1 : ORTHOTOPE</code> <code>ORTHOPE2 : ORTHOTOPE</code> <code>N : int</code>
Restriction(s)	<code> ORTHOPE > 0</code> <code>require_at_least(2, ORTHOTOPE, [ori, siz, end])</code> <code>ORTHOPE.siz ≥ 0</code> <code> ORTHOPE1 = ORTHOPE2 </code> <code>orth_link_ori_siz_end(ORTHOPE1)</code> <code>orth_link_ori_siz_end(ORTHOPE2)</code> <code>N > 0</code> <code>N ≤ ORTHOPE1 </code>
Purpose	
Arc input(s)	<code>ORTHOPE1 ORTHOTOPE2</code>
Arc generator	<code>PRODUCT(=) ↦ collection(orthotope1, orthotope2)</code>
Arc arity	2
Arc constraint(s)	$\bigwedge \left(\begin{array}{l} \text{orthotope1.key} = N, \\ \text{orthotope1.ori} < \text{orthotope2.end}, \\ \text{orthotope2.ori} < \text{orthotope1.end}, \\ \text{orthotope1.siz} > 0, \\ \text{orthotope2.siz} > 0 \end{array} \right) \Rightarrow$ $\bigwedge \left(\begin{array}{l} \min(\text{orthotope1.end}, \text{orthotope2.end}) - \max(\text{orthotope1.ori}, \text{orthotope2.ori}) = \\ \text{orthotope1.siz} \\ \text{orthotope1.siz} = \text{orthotope2.siz} \end{array} , 1 \right)$
Graph property(ies)	<code>NARC = 1</code>
Example	<code>two_orth_column</code> $\left(\left\{ \left\{ \begin{array}{l} \text{ori} - 1 \quad \text{siz} - 3 \quad \text{end} - 4, \\ \text{ori} - 1 \quad \text{siz} - 1 \quad \text{end} - 2 \end{array} \right\}, \right. \right. \\ \left. \left. \left\{ \begin{array}{l} \text{ori} - 4 \quad \text{siz} - 2 \quad \text{end} - 6, \\ \text{ori} - 1 \quad \text{siz} - 3 \quad \text{end} - 4 \end{array} \right\}, 1 \right. \right)$
Used in	<code>diffn_column</code> .
See also	<code>diffn</code> .
Key words	geometrical constraint, positioning constraint, orthotope, guillotine cut.

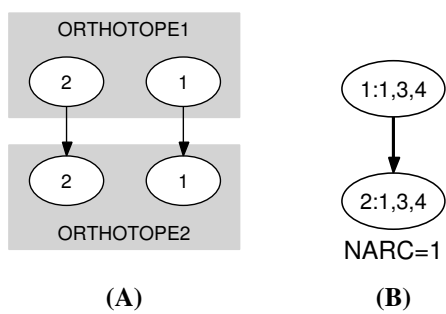


Figure 4.424: Initial and final graph of the two_orth_column constraint

4.227 two_orth_do_not_overlap

Origin	Used for defining diffn.
Constraint	<code>two_orth_do_not_overlap(ORTHOTOPE1, ORTHOTOPE2)</code>
Type(s)	<code>ORTHOTOPE : collection(ori - dvar, siz - dvar, end - dvar)</code>
Argument(s)	<code>ORTHOTOPE1 : ORTHOTOPE</code> <code>ORTHOTOPE2 : ORTHOTOPE</code>
Restriction(s)	<code> ORTHOTOPE > 0</code> <code>require_at_least(2, ORTHOTOPE, [ori, siz, end])</code> <code>ORTHOTOPE.siz ≥ 0</code> <code> ORTHOTOPE1 = ORTHOTOPE2 </code> <code>orth_link_ori_siz_end(ORTHOTOPE1)</code> <code>orth_link_ori_siz_end(ORTHOTOPE2)</code>
Purpose	For two orthotopes O_1 and O_2 enforce that there exist at least one dimension i such that the projections on i of O_1 and O_2 do not overlap.
Arc input(s)	<code>ORTHOTOPE1 ORTHOTOPE2</code>
Arc generator	<code>SYMMETRIC_PRODUCT(=) ↦ collection(orthotope1, orthotope2)</code>
Arc arity	2
Arc constraint(s)	<code>orthotope1.end ≤ orthotope2.ori ∨ orthotope1.siz = 0</code>
Graph property(ies)	$\overline{\text{NARC}} \geq 1$
Example	$\text{two_orth_do_not_overlap} \left(\left(\begin{array}{l} \left\{ \begin{array}{lll} \text{ori} - 2 & \text{siz} - 2 & \text{end} - 4, \\ \text{ori} - 1 & \text{siz} - 3 & \text{end} - 4 \end{array} \right\}, \\ \left\{ \begin{array}{lll} \text{ori} - 4 & \text{siz} - 4 & \text{end} - 8, \\ \text{ori} - 3 & \text{siz} - 3 & \text{end} - 6 \end{array} \right\} \end{array} \right) \right)$ <p>Parts (A) and (B) of Figure 4.425 respectively show the initial and final graph. Since we use the $\overline{\text{NARC}}$ graph property, the unique arc of the final graph is stressed in bold. It corresponds to the fact that the projection in dimension 1 of the first orthotope is located before the projection in dimension 1 of the second orthotope. Therefore the two orthotopes do not overlap.</p>
Graph model	We build an initial graph where each arc corresponds to the fact that, either the projection of an orthotope on a given dimension is empty, either it is located before the projection in the same dimension of the other orthotope. Finally we ask that at least one arc constraint remains in the final graph.

Automaton

Figure 4.426 depicts the automaton associated to the `two_orth_do_not_overlap` constraint. Let $ORI1_i$, $SIZ1_i$ and $END1_i$ respectively be the `ori`, the `siz` and the `end` attributes of the i^{th} item of the `ORTHOTOPE1` collection. Let $ORI2_i$, $SIZ2_i$ and $END2_i$ respectively be the `ori`, the `siz` and the `end` attributes of the i^{th} item of the `ORTHOTOPE2` collection. To each sextuple $(ORI1_i, SIZ1_i, END1_i, ORI2_i, SIZ2_i, END2_i)$ corresponds a 0-1 signature variable S_i as well as the following signature constraint: $((SIZ1_i > 0) \wedge (SIZ2_i > 0) \wedge (END1_i > ORI2_i) \wedge (END2_i > ORI1_i)) \Leftrightarrow S_i$.

Used in

`diffn`.

Key words

geometrical constraint, non-overlapping, orthotope, Berge-acyclic constraint network, automaton, automaton without counters.

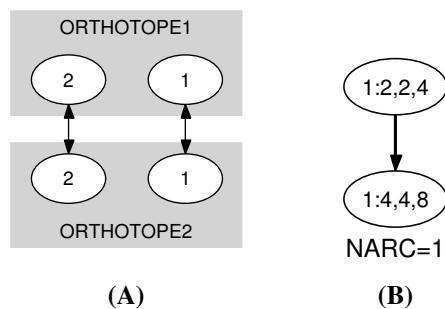
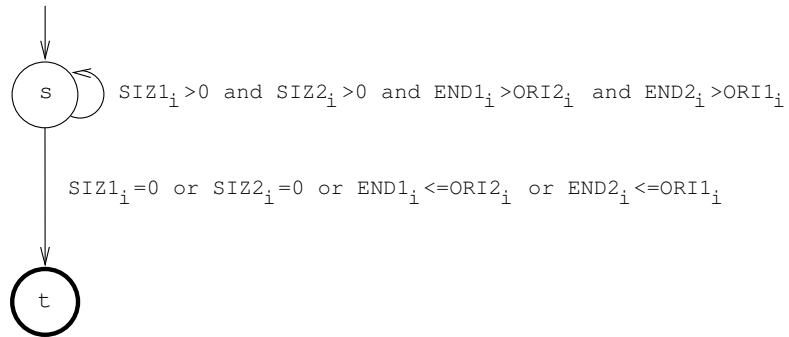
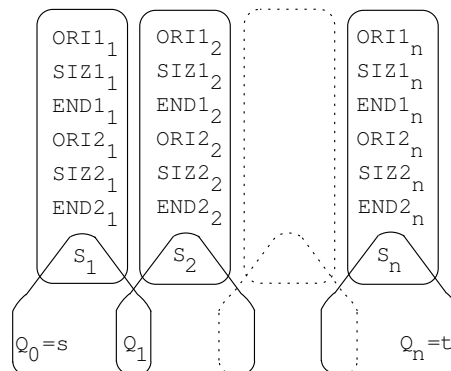



Figure 4.425: Initial and final graph of the `two_orth_do_not_overlap` constraint

Figure 4.426: Automaton of the `two_orth_do_not_overlap` constraintFigure 4.427: Hypergraph of the reformulation corresponding to the automaton of the `two_orth_do_not_overlap` constraint

20030820

927

4.228 two_orth_include

Origin	Used for defining <code>diffn_include</code> .
Constraint	<code>two_orth_include(ORTHOTOPE1, ORTHOTOPE2, N)</code>
Type(s)	ORTHOTOPE : <code>collection(ori - dvar, siz - dvar, end - dvar)</code>
Argument(s)	ORTHOTOPE1 : ORTHOTOPE ORTHOTOPE2 : ORTHOTOPE N : <code>int</code>
Restriction(s)	<code> ORTHOTOPE > 0</code> <code>require_at_least(2, ORTHOTOPE, [ori, siz, end])</code> <code>ORTHOTOPE.siz ≥ 0</code> <code> ORTHOTOPE1 = ORTHOTOPE2 </code> <code>orth_link_ori_siz_end(ORTHOTOPE1)</code> <code>orth_link_ori_siz_end(ORTHOTOPE2)</code> <code>N > 0</code> <code>N ≤ ORTHOTOPE1 </code>
Purpose	
Arc input(s)	ORTHOTOPE1 ORTHOTOPE2
Arc generator	<i>PRODUCT(=)</i> \mapsto <code>collection(orthotope1, orthotope2)</code>
Arc arity	2
Arc constraint(s)	$\bigwedge \left(\begin{array}{l} \text{orthotope1.key} = N, \\ \text{orthotope1.ori} < \text{orthotope2.end}, \\ \text{orthotope2.ori} < \text{orthotope1.end}, \\ \text{orthotope1.siz} > 0, \\ \text{orthotope2.siz} > 0 \end{array} \right) \Rightarrow$ $\bigvee \left(\begin{array}{l} \min(\text{orthotope1.end}, \text{orthotope2.end}) - \max(\text{orthotope1.ori}, \text{orthotope2.ori}) = \\ \text{orthotope1.siz} \\ \min(\text{orthotope1.end}, \text{orthotope2.end}) - \max(\text{orthotope1.ori}, \text{orthotope2.ori}) = \\ \text{orthotope2.siz} \end{array} \right),$
Graph property(ies)	<u>NARC = 1</u>
Example	<code>two_orth_include</code> $\left(\left(\begin{array}{l} \left\{ \text{ori} - 1 \quad \text{siz} - 3 \quad \text{end} - 4, \right\} \\ \left\{ \text{ori} - 1 \quad \text{siz} - 1 \quad \text{end} - 2 \right\}, \\ \left\{ \text{ori} - 1 \quad \text{siz} - 2 \quad \text{end} - 3, \right\} \\ \left\{ \text{ori} - 2 \quad \text{siz} - 3 \quad \text{end} - 5 \right\} \end{array} \right), 1 \right)$
Used in	<code>diffn_include</code> .

See also diffn.

Key words geometrical constraint, positioning constraint, orthotope.

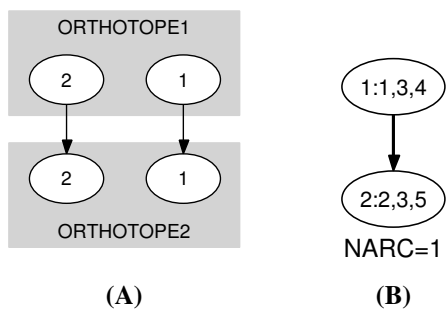


Figure 4.428: Initial and final graph of the two_orth_include constraint

4.229 used_by

Origin	N. Beldiceanu
Constraint	<code>used_by(VARIABLES1, VARIABLES2)</code>
Argument(s)	VARIABLES1 : <code>collection(var - dvar)</code> VARIABLES2 : <code>collection(var - dvar)</code>
Restriction(s)	$ VARIABLES1 \geq VARIABLES2 $ <code>required(VARIABLES1, var)</code> <code>required(VARIABLES2, var)</code>
Purpose	All the values of the variables of collection VARIABLES2 are used by the variables of collection VARIABLES1.
Arc input(s)	VARIABLES1 VARIABLES2
Arc generator	$\text{PRODUCT} \mapsto \text{collection}(\text{variables1}, \text{variables2})$
Arc arity	2
Arc constraint(s)	<code>variables1.var = variables2.var</code>
Graph property(ies)	<ul style="list-style-type: none"> • for all connected components: $\text{NSOURCE} \geq \text{NSINK}$ • $\text{NSINK} = VARIABLES2$

Example	$\text{used_by} \left(\left(\begin{array}{l} \text{var} - 1, \\ \text{var} - 9, \\ \text{var} - 1, \\ \text{var} - 5, \\ \text{var} - 2, \\ \text{var} - 1 \end{array} \right), \left\{ \text{var} - 1, \text{var} - 1, \text{var} - 2, \text{var} - 5 \right\} \right)$
----------------	---

Parts (A) and (B) of Figure 4.429 respectively show the initial and final graph. Since we use the **NSOURCE** and **NSINK** graph properties, the source and sink vertices of the final graph are stressed with a double circle. Since there is a constraint on each connected component of the final graph we also show the different connected components. Each of them corresponds to an equivalence class according to the arc constraint. Note that the vertex corresponding to the variable assigned to value 9 was removed from the final graph since there is no arc for which the associated equality constraint holds. The `used_by` constraint holds since:

- For each connected component of the final graph the number of sources is greater than or equal to the number of sinks.
- The number of sinks of the final graph is equal to $|VARIABLES2|$.

- Signature** Since the initial graph contains only sources and sinks, and since sources of the initial graph cannot become sinks of the final graph, we have that the maximum number of sinks of the final graph is equal to $|\text{VARIABLES2}|$. Therefore we can rewrite $\text{NSINK} = |\text{VARIABLES2}|$ to $\text{NSINK} \geq |\text{VARIABLES2}|$ and simplify $\overline{\text{NSINK}}$ to $\overline{\text{NSINK}}$.
- Automaton** Figure 4.430 depicts the automaton associated to the `used_by` constraint. To each item of the collection `VARIABLES1` corresponds a signature variable S_i , which is equal to 0. To each item of the collection `VARIABLES2` corresponds a signature variable $S_{i+|\text{VARIABLES1}|}$, which is equal to 1.
- Algorithm** As described in [141] we can pad `VARIABLES2` with dummy variables such that its cardinality will be equal to that cardinality of `VARIABLES1`. The domain of a dummy variable contains all of the values. Then, we have a same constraint between the two sets. Direct arc-consistency and bound-consistency algorithms are also proposed in [141] and in [142].
- Key words** constraint between two collections of variables, inclusion, flow, bound-consistency, automaton, automaton with array of counters.

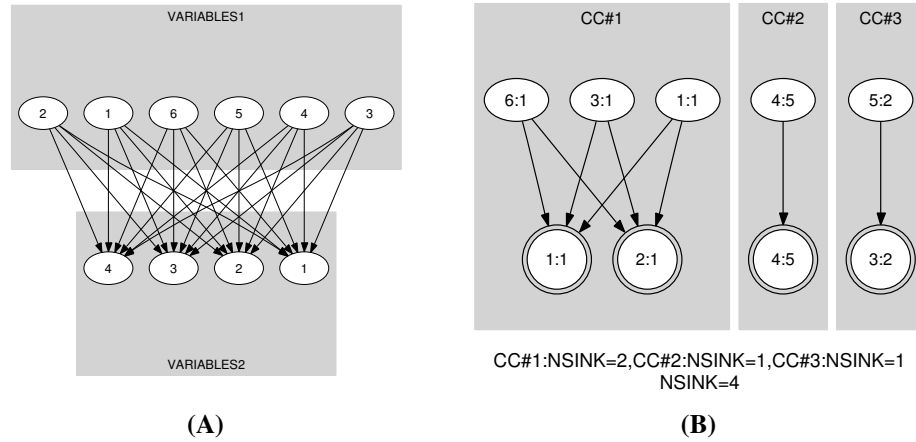


Figure 4.429: Initial and final graph of the used_by constraint

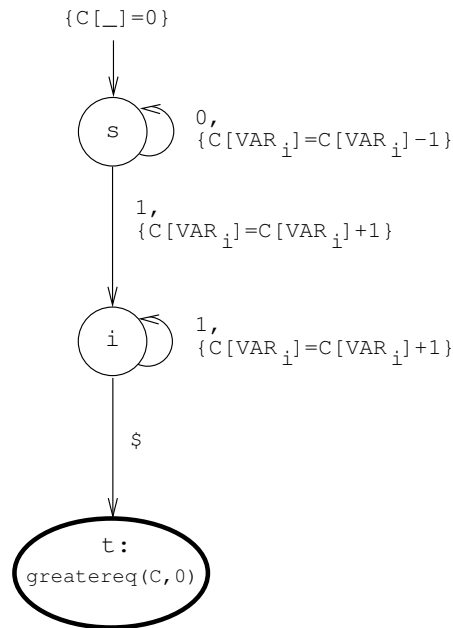


Figure 4.430: Automaton of the used_by constraint

20000128

933

4.230 used_by_interval

Origin	Derived from used_by.
Constraint	used_by_interval(VARIABLES1, VARIABLES2, SIZE_INTERVAL)
Argument(s)	VARIABLES1 : collection(var - dvar) VARIABLES2 : collection(var - dvar) SIZE_INTERVAL : int
Restriction(s)	$ \text{VARIABLES1} \geq \text{VARIABLES2} $ required(VARIABLES1, var) required(VARIABLES2, var) $\text{SIZE_INTERVAL} > 0$
Purpose	Let N_i (respectively M_i) denote the number of variables of the collection VARIABLES1 (respectively VARIABLES2) that take a value in the interval $[\text{SIZE_INTERVAL} \cdot i, \text{SIZE_INTERVAL} \cdot i + \text{SIZE_INTERVAL} - 1]$. For all integer i we have $M_i > 0 \Rightarrow N_i > 0$.
Arc input(s)	VARIABLES1 VARIABLES2
Arc generator	$\text{PRODUCT} \mapsto \text{collection}(\text{variables1}, \text{variables2})$
Arc arity	2
Arc constraint(s)	$\text{variables1.var}/\text{SIZE_INTERVAL} = \text{variables2.var}/\text{SIZE_INTERVAL}$
Graph property(ies)	<ul style="list-style-type: none"> • for all connected components: $\text{NSOURCE} \geq \text{NSINK}$ • $\text{NSINK} = \text{VARIABLES2}$

Example	$\text{used_by_interval} \left(\left(\left(\begin{array}{c} \text{var} - 1, \\ \text{var} - 9, \\ \text{var} - 1, \\ \text{var} - 8, \\ \text{var} - 6, \\ \text{var} - 2 \end{array} \right) \right), \left(\begin{array}{c} \text{var} - 1, \\ \text{var} - 0, \\ \text{var} - 7, \\ \text{var} - 7 \end{array} \right), 3 \right)$
----------------	---

In the previous example, the third parameter SIZE_INTERVAL defines the following family of intervals $[3 \cdot k, 3 \cdot k + 2]$, where k is an integer. Parts (A) and (B) of Figure 4.431 respectively show the initial and final graph. Since we use the NSOURCE and NSINK graph properties, the source and sink vertices of the final graph are stressed with a double circle. Since there is a constraint on each connected component of the final graph we also show the different connected components. Each of them corresponds to an equivalence class according to the arc constraint. Note that the vertex corresponding to the variable that takes value 9 was removed from the final graph since there is no arc for which the associated equivalence constraint holds. The used_by_interval constraint holds since:

- For each connected component of the final graph the number of sources is greater than or equal to the number of sinks.
- The number of sinks of the final graph is equal to $|\text{VARIABLES2}|$.

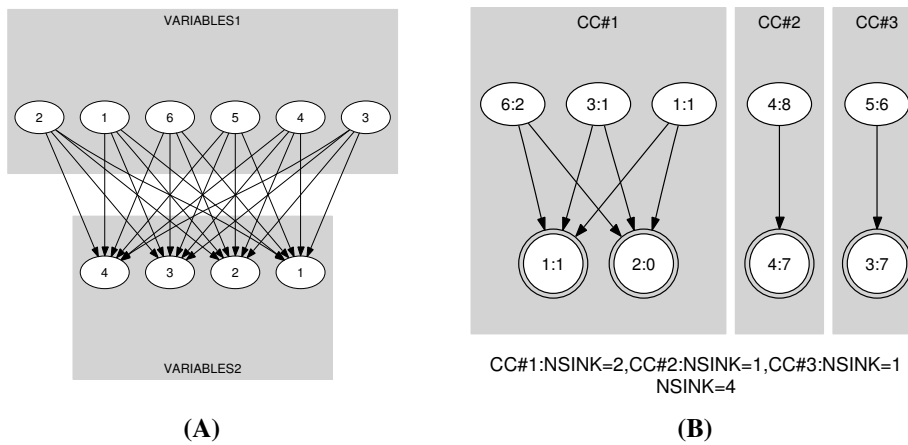


Figure 4.431: Initial and final graph of the `used_by_interval` constraint

Signature

Since the initial graph contains only sources and sinks, and since sources of the initial graph cannot become sinks of the final graph, we have that the maximum number of sinks of the final graph is equal to $|\text{VARIABLES2}|$. Therefore we can rewrite $\text{NSINK} = |\text{VARIABLES2}|$ to $\text{NSINK} \geq |\text{VARIABLES2}|$ and simplify $\overline{\text{NSINK}}$ to NSINK .

See also

`used_by`.

Key words

constraint between two collections of variables, inclusion, interval.

4.231 used_by_modulo

Origin	Derived from used_by.
Constraint	used_by_modulo(VARIABLES1, VARIABLES2, M)
Argument(s)	VARIABLES1 : collection(var - dvar) VARIABLES2 : collection(var - dvar) M : int
Restriction(s)	$ \text{VARIABLES1} \geq \text{VARIABLES2} $ required(VARIABLES1, var) required(VARIABLES2, var) $M > 0$
Purpose	For each integer R in $[0, M - 1]$, let $N1_R$ (respectively $N2_R$) denote the number of variables of VARIABLES1 (respectively VARIABLES2) which have R as a rest when divided by M . For all R in $[0, M - 1]$ we have $N2_R > 0 \Rightarrow N1_R > 0$.
Arc input(s)	VARIABLES1 VARIABLES2
Arc generator	$\text{PRODUCT} \mapsto \text{collection}(\text{variables1}, \text{variables2})$
Arc arity	2
Arc constraint(s)	$\text{variables1.var mod } M = \text{variables2.var mod } M$
Graph property(ies)	<ul style="list-style-type: none"> • for all connected components: $\text{NSOURCE} \geq \text{NSINK}$ • $\text{NSINK} = \text{VARIABLES2}$

Example	$\text{used_by_modulo} \left(\left(\left\{ \begin{array}{l} \text{var} - 1, \\ \text{var} - 9, \\ \text{var} - 4, \\ \text{var} - 5, \\ \text{var} - 2, \\ \text{var} - 1 \end{array} \right\}, \right. \right. \\ \left. \left. \left\{ \begin{array}{l} \text{var} - 7, \\ \text{var} - 1, \\ \text{var} - 2, \\ \text{var} - 5 \end{array} \right\}, 3 \right) \right)$
----------------	--

Parts (A) and (B) of Figure 4.432 respectively show the initial and final graph. Since we use the **NSOURCE** and **NSINK** graph properties, the source and sink vertices of the final graph are stressed with a double circle. Since there is a constraint on each connected component of the final graph we also show the different connected components. Each of them corresponds to an equivalence class according to the arc constraint. Note that the vertex corresponding to the variable that takes value 9 was removed from the final graph since there is no arc for which the associated equivalence constraint holds. The used_by_modulo constraint holds since:

- For each connected component of the final graph the number of sources is greater than or equal to the number of sinks.
- The number of sinks of the final graph is equal to $|\text{VARIABLES2}|$.

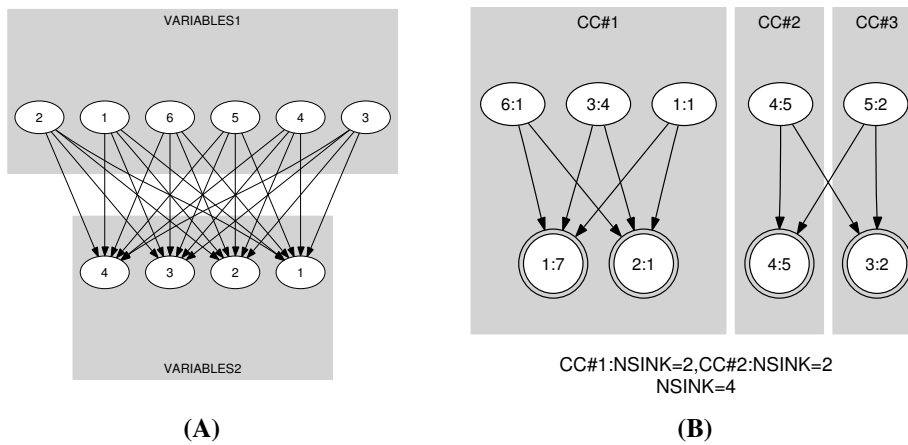


Figure 4.432: Initial and final graph of the `used_by_modulo` constraint

Signature

Since the initial graph contains only sources and sinks, and since sources of the initial graph cannot become sinks of the final graph, we have that the maximum number of sinks of the final graph is equal to $|\text{VARIABLES2}|$. Therefore we can rewrite $\text{NSINK} = |\text{VARIABLES2}|$ to $\text{NSINK} \geq |\text{VARIABLES2}|$ and simplify $\overline{\text{NSINK}}$ to NSINK .

See also

`used_by`.

Key words

constraint between two collections of variables, inclusion, modulo.

4.232 used_by_partition

Origin	Derived from used_by.
Constraint	used_by_partition(VARIABLES1, VARIABLES2, PARTITIONS)
Type(s)	VALUES : collection(val – int)
Argument(s)	VARIABLES1 : collection(var – dvar) VARIABLES2 : collection(var – dvar) PARTITIONS : collection(p – VALUES)
Restriction(s)	required(VALUES, val) distinct(VALUES, val) $ \text{VARIABLES1} \geq \text{VARIABLES2} $ required(VARIABLES1, var) required(VARIABLES2, var) required(PARTITIONS, p) $ \text{PARTITIONS} \geq 2$
Purpose	For each integer i in $[1, \text{PARTITIONS}]$, let $N1_i$ (respectively $N2_i$) denote the number of variables of VARIABLES1 (respectively VARIABLES2) which take their value in the i^{th} partition of the collection PARTITIONS. For all i in $[1, \text{PARTITIONS}]$ we have $N2_i > 0 \Rightarrow N1_i > 0$.
Arc input(s)	VARIABLES1 VARIABLES2
Arc generator	$\text{PRODUCT} \mapsto \text{collection}(\text{variables1}, \text{variables2})$
Arc arity	2
Arc constraint(s)	in_same_partition(variables1.var, variables2.var, PARTITIONS)
Graph property(ies)	<ul style="list-style-type: none"> • for all connected components: $\text{NSOURCE} \geq \text{NSINK}$ • $\text{NSINK} = \text{VARIABLES2}$

Example	$\text{used_by_partition} \left(\left(\left(\begin{array}{l} \text{var} - 1, \\ \text{var} - 9, \\ \text{var} - 1, \\ \text{var} - 6, \\ \text{var} - 2, \\ \text{var} - 3 \end{array} \right) \right), \left(\begin{array}{l} \text{var} - 1, \\ \text{var} - 3, \\ \text{var} - 6, \\ \text{var} - 6 \end{array} \right), \left(\begin{array}{l} \text{p} - \{\text{val} - 1, \text{val} - 3\}, \\ \text{p} - \{\text{val} - 4\}, \\ \text{p} - \{\text{val} - 2, \text{val} - 6\} \end{array} \right) \right)$
----------------	--

Parts (A) and (B) of Figure 4.433 respectively show the initial and final graph. Since we use the **NSOURCE** and **NSINK** graph properties, the source and sink vertices of the final graph are stressed with a double circle. Since there is a constraint on each connected component of the final graph we also show the different connected components. Each of them corresponds to an equivalence class according to the arc constraint. Note that the vertex corresponding to the variable that takes value 9 was removed from the final graph since there is no arc for which the associated equivalence constraint holds. The **used_by_partition** constraint holds since:

- For each connected component of the final graph the number of sources is greater than or equal to the number of sinks.
- The number of sinks of the final graph is equal to $|\text{VARIABLES2}|$.

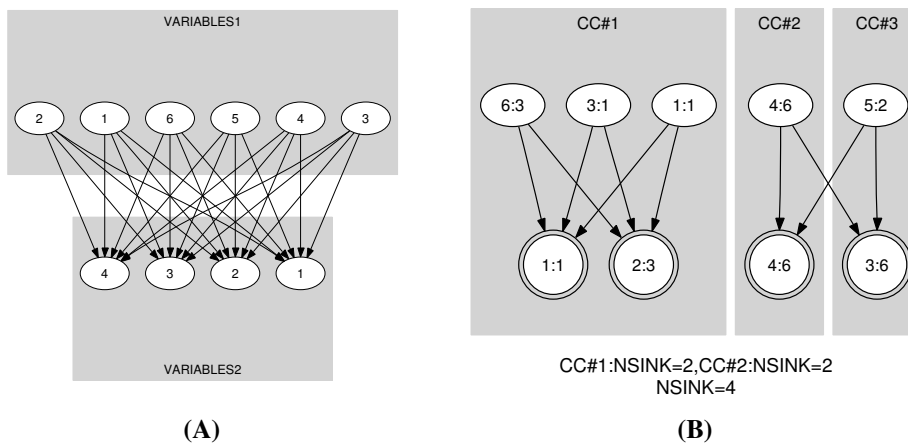


Figure 4.433: Initial and final graph of the **used_by_partition** constraint

Signature

Since the initial graph contains only sources and sinks, and since sources of the initial graph cannot become sinks of the final graph, we have that the maximum number of sinks of the final graph is equal to $|\text{VARIABLES2}|$. Therefore we can rewrite $\text{NSINK} = |\text{VARIABLES2}|$ to $\text{NSINK} \geq |\text{VARIABLES2}|$ and simplify $\overline{\text{NSINK}}$ to NSINK .

See also

used_by, **in_same_partition**.

Key words

constraint between two collections of variables, inclusion, partition.

4.233 valley

Origin Derived from `inflexion`.

Constraint `valley(N, VARIABLES)`

Argument(s) `N` : `dvar`
`VARIABLES` : `collection(var - dvar)`

Restriction(s) $N \geq 0$
 $2 * N \leq \max(|VARIABLES| - 1, 0)$
`required(VARIABLES, var)`

Purpose A variable V_k ($1 < k < m$) of the sequence of variables $VARIABLES = V_1, \dots, V_m$ is a *valley* if and only if there exist an i ($1 < i \leq k$) such that $V_{i-1} > V_i$ and $V_i = V_{i+1} = \dots = V_k$ and $V_k < V_{k+1}$. N is the total number of valleys of the sequence of variables $VARIABLES$.

Example `valley` $\left(1, \left\{ \begin{array}{l} \text{var} - 1, \\ \text{var} - 1, \\ \text{var} - 4, \\ \text{var} - 8, \\ \text{var} - 8, \\ \text{var} - 2, \\ \text{var} - 7, \\ \text{var} - 1 \end{array} \right\} \right)$

The previous constraint holds since the sequence 1 1 4 8 8 2 7 1 contains one valley which corresponds to the variable which is assigned to value 2.

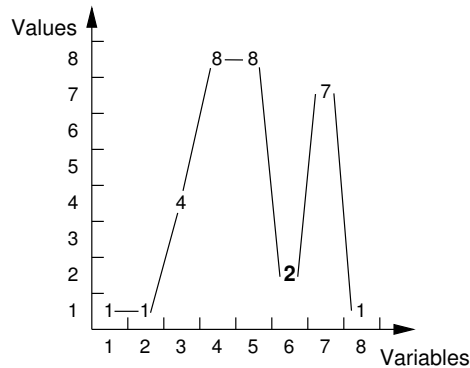


Figure 4.434: The sequence and its unique valley

Automaton Figure 4.435 depicts the automaton associated to the `valley` constraint. To each pair of consecutive variables (VAR_i, VAR_{i+1}) of the collection $VARIABLES$ corresponds a signature variable S_i . The following signature constraint links VAR_i, VAR_{i+1} and S_i : $(VAR_i < VAR_{i+1} \Leftrightarrow S_i = 0) \wedge (VAR_i = VAR_{i+1} \Leftrightarrow S_i = 1) \wedge (VAR_i > VAR_{i+1} \Leftrightarrow S_i = 2)$.

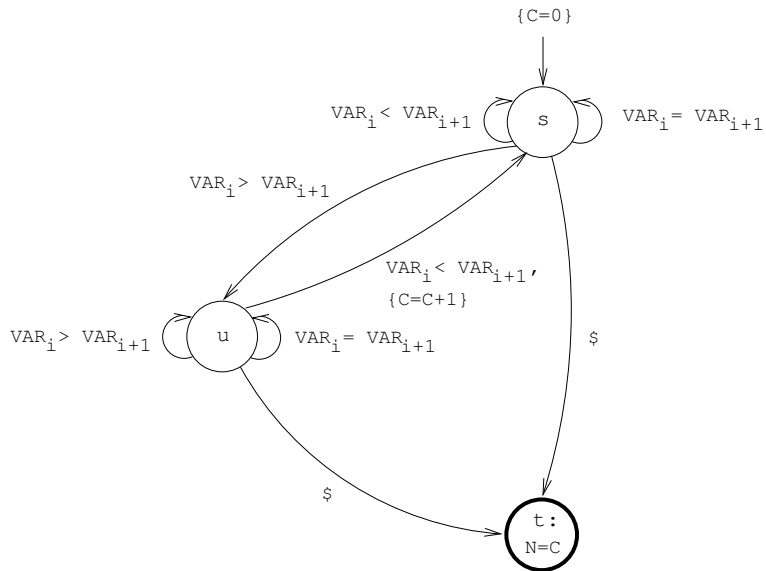


Figure 4.435: Automaton of the valley constraint

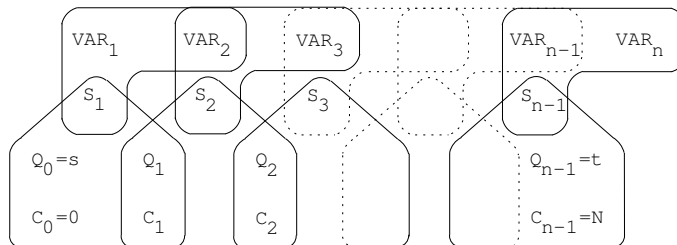


Figure 4.436: Hypergraph of the reformulation corresponding to the automaton of the valley constraint

Usage	Useful for constraining the number of <i>valleys</i> of a sequence of domain variables.
Remark	Since the arity of the arc constraint is not fixed, the <i>valley</i> constraint cannot be currently described. However, this would not hold anymore if we were introducing a slot that specifies how to merge adjacent vertices of the final graph.
See also	<i>no_valley</i> , <i>inflexion</i> , <i>peak</i> .
Key words	sequence, automaton, automaton with counters, sliding cyclic(1) constraint network(2).

20040530

943

4.234 `vec_eq_tuple`

Origin	Used for defining <code>in_relation</code> .
Constraint	<code>vec_eq_tuple(VARIABLES, TUPLE)</code>
Argument(s)	<p><code>VARIABLES</code> : collection(var – dvar)</p> <p><code>TUPLE</code> : collection(val – int)</p>
Restriction(s)	<p><code>required(VARIABLES, var)</code></p> <p><code>required(TUPLE, val)</code></p> <p>$VARIABLES = TUPLE$</p>
Purpose	Enforce a vector of domain variables to be equal to a tuple of values.
Arc input(s)	<code>VARIABLES TUPLE</code>
Arc generator	$PRODUCT(=) \mapsto \text{collection}(\text{variables}, \text{tuple})$
Arc arity	2
Arc constraint(s)	<code>variables.var = tuple.val</code>
Graph property(ies)	$NARC = VARIABLES $
Example	$\text{vec_eq_tuple} \left(\begin{array}{l} \{\text{var} - 5, \text{var} - 3, \text{var} - 3\}, \\ \{\text{val} - 5, \text{val} - 3, \text{val} - 3\} \end{array} \right)$

Parts (A) and (B) of Figure 4.437 respectively show the initial and final graph. Since we use the **NARC** graph property, the arcs of the final graph are stressed in bold.

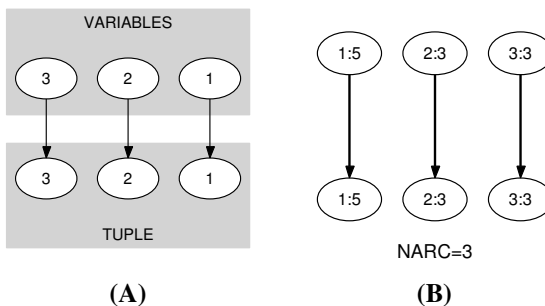


Figure 4.437: Initial and final graph of the `vec_eq_tuple` constraint

Signature

Since we use the arc generator $PRODUCT(=)$ on the collections `VARIABLES` and `TUPLE`, and because of the restriction $|VARIABLES| = |TUPLE|$, the maximum number of arcs of the final graph is equal to $|VARIABLES|$. Therefore we can rewrite the graph property $NARC = |VARIABLES|$ to $NARC \geq |VARIABLES|$ and simplify NARC to NARC.

20030820

945

Used in in_relation.

Key words value constraint, tuple.

4.235 weighted_partial_alldiff

Origin	[160, page 71]
Constraint	<code>weighted_partial_alldiff(VARIABLES, UNDEFINED, VALUES, COST)</code>
Synonym(s)	<code>weighted_partial_alldifferent</code> , <code>weighted_partial_alldistinct</code> , <code>wpa</code> .
Argument(s)	<p><code>VARIABLES</code> : <code>collection(var – dvar)</code> <code>UNDEFINED</code> : <code>int</code> <code>VALUES</code> : <code>collection(val – int, weight – int)</code> <code>COST</code> : <code>dvar</code></p>
Restriction(s)	<p><code>required(VARIABLES, var)</code> <code>required(VALUES, [val, weight])</code> <code>in_attr(VARIABLES, var, VALUES, val)</code> <code>distinct(VALUES, val)</code></p>

Purpose

All variables of the `VARIABLES` collection which are not assigned to value `UNDEFINED` must have pairwise distinct values from the `val` attribute of the `VALUES` collection. In addition `COST` is the sum of the `weight` attributes associated to the values assigned to the variables of `VARIABLES`. Within the `VALUES` collection, value `UNDEFINED` must be explicitly defined with a weight of 0.

Arc input(s)	<code>VARIABLES VALUES</code>
Arc generator	<code>PRODUCT</code> \mapsto <code>collection(variables, values)</code>
Arc arity	2
Arc constraint(s)	<ul style="list-style-type: none"> • <code>variables.var</code> \neq <code>UNDEFINED</code> • <code>variables.var</code> = <code>values.val</code>
Graph property(ies)	<ul style="list-style-type: none"> • <code>MAX_ID</code> \leq 1 • <code>SUM(VALUES, weight)</code> = <code>COST</code>

Example

`weighted_partial_alldiff` $\left(\left(\left(\begin{array}{l} \text{var - 4,} \\ \text{var - 0,} \\ \text{var - 1,} \\ \text{var - 2,} \\ \text{var - 0,} \\ \text{var - 0} \end{array} \right) \right), 0, \left(\begin{array}{ll} \text{val - 0} & \text{weight - 0,} \\ \text{val - 1} & \text{weight - 2,} \\ \text{val - 2} & \text{weight - -1,} \\ \text{val - 4} & \text{weight - 7,} \\ \text{val - 5} & \text{weight - -8,} \\ \text{val - 6} & \text{weight - 2} \end{array} \right) \right), 8$

Parts (A) and (B) of Figure 4.438 respectively show the initial and final graph. Since we also use the **SUM** graph property we show the vertices of the final graph from which we compute the total cost in a box. The `weighted_partial_alldiff` constraint holds since no value, except for value `UNDEFINED = 0`, is used more than once and `COST = 8` is equal to the sum of the weights 2, -1 and 7 of the values 1, 2 and 4 assigned to the variables of `VARIABLES`.

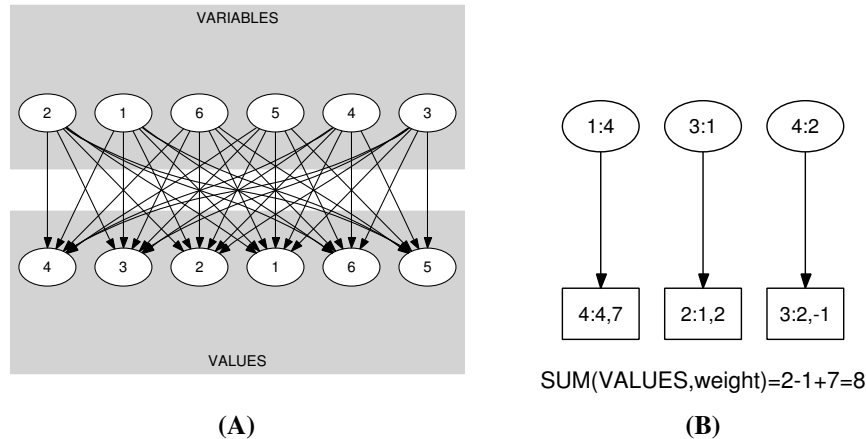


Figure 4.438: Initial and final graph of the `weighted_partial_alldiff` constraint

Graph model

The restriction `in_attr(VARIABLES, var, VALUES, val)` imposes all variables of the `VARIABLES` collection to take a value from the `val` attribute of the `VALUES` collection. We use the *PRODUCT* to generate an arc from every variables of the `VARIABLES` collection to every value of the `VALUES` collection. Because of the arc constraint, the final graph contains only those arcs arriving at a value different from `UNDEFINED`. The graph property `MAX_ID ≤ 1` enforces that no vertex of the final graph has more than one predecessor. As a consequence, all variables of the `VARIABLES` collection which are not assigned to value `UNDEFINED` must have pairwise distinct values.

Usage

In his PhD thesis [160, pages 71–72], Sven Thiel describes the following three potential scenarios of the `weighted_partial_alldiff` constraint:

- Given a set of tasks (i.e. the items of the `VARIABLES` collection), assign to each task a resource (i.e. an item of the `VALUES` collection). Except for the resource associated to value `UNDEFINED`, every resource can be used at most once. The cost of a resource is independent from the task to which the resource is assigned. The cost of value `UNDEFINED` is equal to 0. The total cost `COST` of an assignment corresponds to the sum of the costs of the resources effectively assigned to the tasks. Finally we impose an upper bound on the total cost.
- Given a set of persons (i.e. the items of the `VARIABLES` collection), select for each person an offer (i.e. an item of the `VALUES` collection). Except for the offer associated to value `UNDEFINED`, every offer should be selected at most once. The profit associated to an offer is independent from the person which select that offer. The profit of value `UNDEFINED` is equal to 0. The total benefit `COST` is equal to the sum

of the profits of the offers effectively selected. In addition we impose a lower bound on the total benefit.

- The last scenario deals with an application to an over-constraint problem involving the `alldifferent` constraint. Allowing some variables to take an "undefined" value is done by setting all weights of all the values different from `UNDEFINED` to 1. As a consequence all variables assigned to a value different from `UNDEFINED` will have to take distinct values. The `COST` variable allows to control the number of such variables.

Algorithm

A filtering algorithm is given in [160, pages 73–104]. After showing that, deciding whether the `weighted_partial_alldiff` has a solution is NP-complete, [160, pages 105–106] gives the following results of his filtering algorithm with respect to consistency under the three scenarios previously described:

- For scenario 1, if there is no restriction of the lower bound of the `COST` variable, the filtering algorithm achieves arc-consistency for all variables of the `VARIABLES` collection (but not for the `COST` variable itself).
- For scenario 2, if there is no restriction of the upper bound of the `COST` variable, the filtering algorithm achieves arc-consistency for all variables of the `VARIABLES` collection (but not for the `COST` variable itself).
- Finally, for scenario 3, the filtering algorithm achieves arc-consistency for all variables of the `VARIABLES` collection as well as for the `COST` variable.

See also

`alldifferent`, `alldifferent_except_0`, `minimum_weight_alldifferent`,
`global_cardinality_with_costs`, `soft_alldifferent_var`,
`sum_of_weights_of_distinct_values`.

Key words

cost filtering constraint, soft constraint, all different, assignment, relaxation, joker value, weighted assignment.

Appendix A

Legend for the description

This section provides the list of *restrictions*, of *arc generators*, of *graph generators* and of *set generators* sorted in alphabetic order with the page where there are defined.

Restrictions :

- *Term₁ Comparison Term₂* p. 9
- *distinct* p. 7
- *in_attr* p. 6
- *in_list* p. 6
- *increasing_seq* p. 7
- *required* p. 8
- *require_at_least* p. 8
- *same_size* p. 9

Arc generators :

- *CHAIN* p. 27
- *CIRCUIT* p. 27
- *CLIQUE* p. 27
- *CLIQUE(C)* p. 28
- *GRID* p. 28
- *LOOP* p. 28
- *PATH* p. 28
- *PATH_1* p. 28
- *PATH_N* p. 29
- *PRODUCT* p. 29
- *PRODUCT(C)* p. 29
- *SELF* p. 29
- *SYMMETRIC_PRODUCT* p. 29
- *SYMMETRIC_PRODUCT(C)* p. 29
- *VOID* p. 29

Graph characteristics :

- *DISTANCE* p. 42
- *MAX_DRG* p. 34
- *MAX_ID* p. 34
- *MAX_NCC* p. 34
- *MAX_NSCC* p. 35
- *MAX_OD* p. 35
- *MIN_DRG* p. 35
- *MIN_ID* p. 35
- *MIN_NCC* p. 35
- *MIN_NSCC* p. 36
- *MIN_OD* p. 36
- *NARC* p. 36
- *NARC.NO_LOOP* p. 36
- *NCC* p. 37
- *NSCC* p. 37
- *NSINK* p. 37
- *NSINK.NSOURCE* p. 37
- *NSOURCE* p. 38
- *NTREE* p. 38
- *NVERTEX* p. 38
- *RANGE_DRG* p. 38
- *RANGE_NCC* p. 39
- *RANGE_NSCC* p. 39
- *ORDER* p. 39
- *PATH_FROM_TO* p. 39
- *PRODUCT* p. 40
- *RANGE* p. 40
- *SUM* p. 41
- *SUM.WEIGHT_ARC* p. 42

Set generators :

- *ALL_VERTICES* p. 47
- *CC* p. 47
- *PATH_LENGTH* p. 48
- *PRED* p. 48
- *SUCC* p. 48

Appendix B

Electronic constraint catalog

Contents

B.1	all_differ_from_at_least_k_pos	957
B.2	all_min_dist	958
B.3	alldifferent	959
B.4	alldifferent_between_sets	960
B.5	alldifferent_except_0	961
B.6	alldifferent_interval	962
B.7	alldifferent_modulo	963
B.8	alldifferent_on_intersection	964
B.9	alldifferent_partition	965
B.10	alldifferent_same_value	967
B.11	allperm	968
B.12	among	969
B.13	among_diff_0	971
B.14	among_interval	973
B.15	among_low_up	975
B.16	among_modulo	977
B.17	among_seq	979
B.18	arith	981
B.19	arith_or	983
B.20	arith_sliding	985
B.21	assign_and_counts	989
B.22	assign_and_nvalues	991
B.23	atleast	993
B.24	atmost	995
B.25	balance	996
B.26	balance_interval	997
B.27	balance_modulo	998

B.28	balance_partition	999
B.29	bin_packing	1000
B.30	binary_tree	1001
B.31	cardinality_atleast	1002
B.32	cardinality_atmost	1003
B.33	cardinality_atmost_partition	1004
B.34	change	1005
B.35	change_continuity	1007
B.36	change_pair	1012
B.37	change_partition	1018
B.38	circuit	1020
B.39	circuit_cluster	1021
B.40	circular_change	1023
B.41	clique	1025
B.42	colored_matrix	1026
B.43	coloured_cumulative	1028
B.44	coloured_cumulatives	1030
B.45	common	1032
B.46	common_interval	1033
B.47	common_modulo	1034
B.48	common_partition	1035
B.49	connect_points	1037
B.50	correspondence	1040
B.51	count	1042
B.52	counts	1044
B.53	crossing	1046
B.54	cumulative	1048
B.55	cumulative_product	1050
B.56	cumulative_two_d	1052
B.57	cumulative_with_level_of_priority	1055
B.58	cumulatives	1057
B.59	cutset	1059
B.60	cycle	1060
B.61	cycle_card_on_path	1061
B.62	cycle_or_accessibility	1063
B.63	cycle_resource	1065
B.64	cyclic_change	1067
B.65	cyclic_change_joker	1069
B.66	decreasing	1072
B.67	deepest_valley	1073
B.68	derangement	1075
B.69	differ_from_at_least_k_pos	1076
B.70	diffn	1078

B.71	diffn_column	1080
B.72	diffn_include	1081
B.73	discrepancy	1082
B.74	disjoint	1083
B.75	disjoint_tasks	1084
B.76	disjunctive	1086
B.77	distance_between	1087
B.78	distance_change	1088
B.79	domain_constraint	1091
B.80	elem	1093
B.81	element	1095
B.82	element_greatereq	1097
B.83	element_lesseq	1099
B.84	element_matrix	1101
B.85	element_sparse	1104
B.86	elements	1106
B.87	elements_alldifferent	1107
B.88	elements_sparse	1109
B.89	eq_set	1111
B.90	exactly	1112
B.91	global_cardinality	1114
B.92	global_cardinality_low_up	1115
B.93	global_cardinality_with_costs	1116
B.94	global_contiguity	1118
B.95	golomb	1120
B.96	graph_crossing	1121
B.97	group	1123
B.98	group_skip_isolated_item	1128
B.99	highest_peak	1132
B.100	in	1134
B.101	lin_relation	1136
B.102	in_same_partition	1138
B.103	in_set	1140
B.104	increasing	1141
B.105	indexed_sum	1142
B.106	inflexion	1143
B.107	int_value_precede	1145
B.108	int_value_precede_chain	1146
B.109	interval_and_count	1147
B.110	interval_and_sum	1149
B.111	inverse	1150
B.112	inverse_set	1151
B.113	th_pos_different_from_0	1153

B.114	k_cut	1155
B.115	lex2	1156
B.116	lex_alldifferent	1157
B.117	lex_between	1158
B.118	lex_chain_less	1161
B.119	lex_chain_lesseq	1162
B.120	lex_different	1163
B.121	lex_greater	1165
B.122	lex_greatereq	1167
B.123	lex_less	1169
B.124	lex_lesseq	1171
B.125	link_set_to_booleans	1173
B.126	longest_change	1174
B.127	map	1176
B.128	max_index	1177
B.129	max_n	1179
B.130	max_nvalue	1180
B.131	max_size_set_of_consecutive_var	1181
B.132	maximum	1182
B.133	maximum_modulo	1184
B.134	min_index	1185
B.135	min_n	1187
B.136	min_nvalue	1188
B.137	min_size_set_of_consecutive_var	1189
B.138	minimum	1190
B.139	minimum_except_0	1192
B.140	minimum_greater_than	1194
B.141	minimum_modulo	1196
B.142	minimum_weight_alldifferent	1197
B.143	nclass	1199
B.144	equivalence	1200
B.145	next_element	1201
B.146	next_greater_element	1204
B.147	ninterval	1205
B.148	no_peak	1206
B.149	no_valley	1208
B.150	not_all_equal	1210
B.151	not_in	1212
B.152	npair	1214
B.153	nset_of_consecutive_values	1215
B.154	nvalue	1216
B.155	nvalue_on_intersection	1217
B.156	nvalues	1218

B.157	nvalues_except_0	1219
B.158	one_tree	1220
B.159	orchard	1222
B.160	orth_link_ori_siz_end	1223
B.161	orth_on_the_ground	1224
B.162	orth_on_top_of_orth	1225
B.163	orths_are_connected	1227
B.164	path_from_to	1229
B.165	pattern	1231
B.166	peak	1232
B.167	period	1234
B.168	period_except_0	1235
B.169	place_in_pyramid	1236
B.170	polyomino	1238
B.171	product_ctr	1240
B.172	range_ctr	1241
B.173	relaxed_sliding_sum	1242
B.174	same	1244
B.175	same_and_global_cardinality	1245
B.176	same_intersection	1247
B.177	same_interval	1248
B.178	same_modulo	1249
B.179	same_partition	1250
B.180	sequence_folding	1251
B.181	set_value_precede	1253
B.182	shift	1254
B.183	size_maximal_sequence_alldifferent	1256
B.184	size_maximal_starting_sequence_alldifferent	1257
B.185	sliding_card_skip0	1258
B.186	sliding_distribution	1260
B.187	sliding_sum	1262
B.188	sliding_time_window	1263
B.189	sliding_time_window_from_start	1264
B.190	sliding_time_window_sum	1266
B.191	smooth	1268
B.192	soft_alldifferent_ctr	1270
B.193	soft_alldifferent_var	1271
B.194	soft_same_interval_var	1272
B.195	soft_same_modulo_var	1273
B.196	soft_same_partition_var	1274
B.197	soft_same_var	1276
B.198	soft_used_by_interval_var	1277
B.199	soft_used_by_modulo_var	1278

B.200	soft_used_by_partition_var	1279
B.201	soft_used_by_var	1281
B.202	sort	1282
B.203	sort_permutation	1283
B.204	stage_element	1285
B.205	stretch_circuit	1287
B.206	stretch_path	1289
B.207	strict_lex2	1291
B.208	strictly_decreasing	1292
B.209	strictly_increasing	1294
B.210	strongly_connected	1296
B.211	sum	1297
B.212	sum_ctr	1298
B.213	sum_of_weights_of_distinct_values	1299
B.214	sum_set	1300
B.215	symmetric_alldifferent	1301
B.216	symmetric_cardinality	1302
B.217	symmetric_gcc	1304
B.218	temporal_path	1306
B.219	tour	1308
B.220	track	1310
B.221	tree	1312
B.222	tree_range	1313
B.223	tree_resource	1314
B.224	two_layer_edge_crossing	1316
B.225	two_orth_are_in_contact	1318
B.226	two_orth_column	1320
B.227	two_orth_do_not_overlap	1322
B.228	two_orth_include	1324
B.229	used_by	1326
B.230	used_by_interval	1327
B.231	used_by_modulo	1328
B.232	used_by_partition	1329
B.233	valley	1330
B.234	vec_eq_tuple	1332
B.235	weighted_partial_alldiff	1333

B.1 all_differ_from_at_least_k_pos

```

ctr_date(
  all_differ_from_at_least_k_pos,
  ['20030820', '20040530']).

ctr_origin(
  all_differ_from_at_least_k_pos,
  'Inspired by \\cite{Frutos97}.',
  []).

ctr_types(
  all_differ_from_at_least_k_pos,
  ['VECTOR'-collection(var-dvar)]).

ctr_arguments(
  all_differ_from_at_least_k_pos,
  ['K'-int, 'VECTORS'-collection(vec-'VECTOR')]).

ctr_restrictions(
  all_differ_from_at_least_k_pos,
  [required('VECTOR', var),
   'K'>=0,
   required('VECTORS', vec),
   same_size('VECTORS', vec)]).

ctr_graph(
  all_differ_from_at_least_k_pos,
  ['VECTORS'],
  2,
  ['CLIQUE' (=\\=)>>collection(vectors1, vectors2)],
  [differ_from_at_least_k_pos(
    'K',
    vectors1^vec,
    vectors2^vec)],
  ['NARC'=size('VECTORS')*size('VECTORS')-size('VECTORS')]).

ctr_example(
  all_differ_from_at_least_k_pos,
  all_differ_from_at_least_k_pos(
    2,
    [[vec-[[var-2], [var-5], [var-2], [var-0]]],
     [vec-[[var-3], [var-6], [var-2], [var-1]]],
     [vec-[[var-3], [var-6], [var-1], [var-0]]]]).

```


B.2 all_min_dist

```

ctr_date(all_min_dist,['20050508']).

ctr_origin(all_min_dist,'\cite{Regin97}',[]).

ctr_synonyms(all_min_dist,[minimum_distance]).

ctr_arguments(
    all_min_dist,
    ['MINDIST'-int,'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    all_min_dist,
    ['MINDIST'>0,required('VARIABLES',var),'VARIABLES'^var>=0]).

ctr_graph(
    all_min_dist,
    ['VARIABLES'],
    2,
    ['CLIQUE' (<)>>collection(variables1,variables2)],
    [abs(variables1^var-variables2^var)>='MINDIST'],
    ['NARC'=size('VARIABLES')*(size('VARIABLES')-1)/2]).

ctr_example(
    all_min_dist,
    all_min_dist(2,[[var-5],[var-1],[var-9],[var-3]])).

```

B.3 alldifferent

```

ctr_date(alldifferent, ['20000128', '20030820', '20040530']).
ctr_origin(alldifferent, '\\cite{Lauriere78}', []).
ctr_synonyms(alldifferent, [alldiff, alldistinct]).
ctr_arguments(alldifferent, ['VARIABLES'-collection(var-dvar)]).
ctr_restrictions(alldifferent, [required('VARIABLES', var)]).

ctr_graph(
    alldifferent,
    ['VARIABLES'],
    2,
    ['CLIQUE'>>collection(variables1, variables2)],
    [variables1^var=variables2^var],
    ['MAX_NSCC'=<1]).

ctr_example(
    alldifferent,
    alldifferent([[var-5], [var-1], [var-9], [var-3]])).

```

B.4 alldifferent_between_sets

```
ctr_date(alldifferent_between_sets, ['20030820']).  
  
ctr_origin(alldifferent_between_sets, 'ILOG', []).  
  
ctr_synonyms(  
    alldifferent_between_sets,  
    [all_null_intersect,  
     alldiff_between_sets,  
     alldistinct_between_sets]).  
  
ctr_arguments(  
    alldifferent_between_sets,  
    ['VARIABLES'-collection(var-svar)]).  
  
ctr_restrictions(  
    alldifferent_between_sets,  
    [required('VARIABLES', var)]).  
  
ctr_graph(  
    alldifferent_between_sets,  
    ['VARIABLES'],  
    2,  
    ['CLIQUE'>>collection(variables1, variables2)],  
    [eq_set(variables1^var, variables2^var)],  
    ['MAX_NSCC'=<1]).  
  
ctr_example(  
    alldifferent_between_sets,  
    alldifferent_between_sets(  
        [[var-{3,5}], [var-{}], [var-{3}], [var-{3,5,7}]])).
```

B.5 alldifferent_except_0

```

ctr_date(
    alldifferent_except_0,
    ['20000128', '20030820', '20040530']).

ctr_origin(
    alldifferent_except_0,
    'Derived from %c.',
    [alldifferent]).

ctr_synonyms(
    alldifferent_except_0,
    [alldiff_except_0, alldistinct_except_0]).

ctr_arguments(
    alldifferent_except_0,
    ['VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    alldifferent_except_0,
    [required('VARIABLES', var)]).

ctr_graph(
    alldifferent_except_0,
    ['VARIABLES'],
    2,
    ['CLIQUE' >> collection(variables1, variables2)],
    [variables1^var=\=0, variables1^var=variables2^var],
    ['MAX_NSCC' =<1]).

ctr_example(
    alldifferent_except_0,
    alldifferent_except_0(
        [[var-5], [var-0], [var-1], [var-9], [var-0], [var-3]])).

```

B.6 alldifferent_interval

```

ctr_date(alldifferent_interval, ['20030820']).

ctr_origin(
    alldifferent_interval,
    'Derived from %c.',
    [alldifferent]).

ctr_synonyms(
    alldifferent_interval,
    [alldiff_interval, alldistinct_interval]).

ctr_arguments(
    alldifferent_interval,
    ['VARIABLES'-collection(var-dvar), 'SIZE_INTERVAL'-int]).

ctr_restrictions(
    alldifferent_interval,
    [required('VARIABLES', var), 'SIZE_INTERVAL'>0]).

ctr_graph(
    alldifferent_interval,
    ['VARIABLES'],
    2,
    ['CLIQUE'>>collection(variables1, variables2)],
    [(variables1^var/'SIZE_INTERVAL',
      variables2^var/'SIZE_INTERVAL')],
    ['MAX_NSCC'=<1]).

ctr_example(
    alldifferent_interval,
    alldifferent_interval([[var-2], [var-3], [var-10]], 3)).

```

B.7 alldifferent_modulo

```

ctr_date(alldifferent_modulo,['20030820']).

ctr_origin(
    alldifferent_modulo,
    'Derived from %c.',
    [alldifferent]).

ctr_synonyms(
    alldifferent_modulo,
    [alldiff_modulo,alldistinct_modulo]).

ctr_arguments(
    alldifferent_modulo,
    ['VARIABLES'-collection(var-dvar),'M'-int]).

ctr_restrictions(
    alldifferent_modulo,
    [required('VARIABLES',var),'M'=\=0,'M'>=size('VARIABLES')]).

ctr_graph(
    alldifferent_modulo,
    ['VARIABLES'],
    2,
    ['CLIQUE'>>collection(variables1,variables2)],
    [variables1^var mod 'M'=variables2^var mod 'M'],
    ['MAX_NSCC'=<1]).

ctr_example(
    alldifferent_modulo,
    alldifferent_modulo([[var-25],[var-1],[var-14],[var-3]],5)).

```

B.8 alldifferent_on_intersection

```

ctr_date(alldifferent_on_intersection, ['20040530']).

ctr_origin(
    alldifferent_on_intersection,
    'Derived from %c and %c.',
    [common, alldifferent]).

ctr_synonyms(
    alldifferent_on_intersection,
    [alldiff_on_intersection, alldistinct_on_intersection]).

ctr_arguments(
    alldifferent_on_intersection,
    ['VARIABLES1'-collection(var-dvar),
     'VARIABLES2'-collection(var-dvar)]).

ctr_restrictions(
    alldifferent_on_intersection,
    [required('VARIABLES1', var), required('VARIABLES2', var)]).

ctr_graph(
    alldifferent_on_intersection,
    ['VARIABLES1', 'VARIABLES2'],
    2,
    ['PRODUCT' >> collection(variables1, variables2)],
    [variables1^var=variables2^var],
    ['MAX_NCC' =<2]).

ctr_example(
    alldifferent_on_intersection,
    alldifferent_on_intersection(
        [[var-5], [var-9], [var-1], [var-5]],
        [[var-2], [var-1], [var-6], [var-9], [var-6], [var-2]])).

```

B.9 alldifferent_partition

```

ctr_date(alldifferent_partition, ['20030820']).

ctr_origin(
    alldifferent_partition,
    'Derived from %c.',
    [alldifferent]).

ctr_synonyms(
    alldifferent_partition,
    [alldiff_partition, alldistinct_partition]).

ctr_types(
    alldifferent_partition,
    ['VALUES'-collection(val-int)]).

ctr_arguments(
    alldifferent_partition,
    ['VARIABLES'-collection(var-dvar),
     'PARTITIONS'-collection(p-'VALUES')]).

ctr_restrictions(
    alldifferent_partition,
    [required('VALUES', val),
     distinct('VALUES', val),
     size('VARIABLES')=<size('PARTITIONS'),
     required('VARIABLES', var),
     size('PARTITIONS')>=2,
     required('PARTITIONS', p)]).

ctr_graph(
    alldifferent_partition,
    ['VARIABLES'],
    2,
    ['CLIQUE'>>collection(variables1, variables2)],
    [in_same_partition(
        variables1^var,
        variables2^var,
        'PARTITIONS')],
    ['MAX_NSCC'=<1]).

ctr_example(
    alldifferent_partition,
    alldifferent_partition(
        [[var-6], [var-3], [var-4]],

```



```
[[p-[[val-1],[val-3]]],  
 [p-[[val-4]]],  
 [p-[[val-2],[val-6]]]])).
```

B.10 alldifferent_same_value

```

ctr_date(alldifferent_same_value, ['20000128', '20030820']).

ctr_origin(
    alldifferent_same_value,
    'Derived from %c.',
    [alldifferent]).

ctr_synonyms(
    alldifferent_same_value,
    [alldiff_same_value, alldistinct_same_value]).

ctr_arguments(
    alldifferent_same_value,
    ['NSAME'-dvar,
     'VARIABLES1'-collection(var-dvar),
     'VARIABLES2'-collection(var-dvar)]).

ctr_restrictions(
    alldifferent_same_value,
    ['NSAME'>=0,
     'NSAME'=<size('VARIABLES1'),
     size('VARIABLES1')=size('VARIABLES2'),
     required('VARIABLES1', var),
     required('VARIABLES2', var)]).

ctr_graph(
    alldifferent_same_value,
    ['VARIABLES1', 'VARIABLES2'],
    2,
    [>>('PRODUCT'('CLIQUE', 'LOOP', =),
         collection(variables1, variables2))],
    [variables1^var=variables2^var],
    ['MAX_NSCC'=<1, 'NARC_NO_LOOP'='NSAME']).

ctr_example(
    alldifferent_same_value,
    alldifferent_same_value(
        2,
        [[var-7], [var-3], [var-1], [var-5]],
        [[var-1], [var-3], [var-1], [var-7]])).

```

B.11 allperm

```
ctr_predefined(allperm).  
  
ctr_date(allperm, ['20031008']).  
  
ctr_origin(allperm, '\\cite{FrischJeffersonMiguel03}', []).  
  
ctr_types(allperm, ['VECTOR'-collection(var-dvar)]).  
  
ctr_arguments(allperm, ['MATRIX'-collection(vec-'VECTOR')]).  
  
ctr_restrictions(  
  allperm,  
  [required('VECTOR', var),  
   required('MATRIX', vec),  
   same_size('MATRIX', vec)]).  
  
ctr_example(  
  allperm,  
  allperm(  
    [[vec-[[var-1], [var-2], [var-3]]],  
     [vec-[[var-3], [var-1], [var-2]]]]).  
)
```

B.12 among

```

ctr_automaton(among, among) .

ctr_date(among, ['20000128', '20030820', '20040807']) .

ctr_origin(among, '\\cite{BeldiceanuContejean94}', []).

ctr_arguments(
  among,
  ['NVAR'-dvar,
   'VARIABLES'-collection(var-dvar),
   'VALUES'-collection(val-int)]).

ctr_restrictions(
  among,
  ['NVAR'>=0,
   'NVAR'=<size('VARIABLES'),
   required('VARIABLES', var),
   required('VALUES', val),
   distinct('VALUES', val)]).

ctr_graph(
  among,
  ['VARIABLES'],
  1,
  ['SELF'>>collection(variables)],
  [in(variables^var, 'VALUES')],
  ['NARC'='NVAR']).

ctr_example(
  among,
  among(
    3,
    [[var-4], [var-5], [var-5], [var-4], [var-1]],
    [[val-1], [val-5], [val-8]])).

among(A, B, C) :-
  col_to_list(C, D),
  list_to_fdset(D, E),
  among_signature(B, F, E),
  automaton(
    F,
    G,
    F,
    0..1,

```

```
[source(s), sink(t)],  
[arc(s, 0, s), arc(s, 1, s, [H+1]), arc(s, $, t)],  
[H],  
[0],  
[A]).
```

```
among_signature([], [], A).
```

```
among_signature([[var-A] | B], [C | D], E) :-  
    in_set(A, E) #<=> C,  
    among_signature(B, D, E).
```

B.13 among_diff_0

```

ctr_automaton(among_diff_0,among_diff_0).

ctr_date(among_diff_0,['20040807']).

ctr_origin(
    among_diff_0,
    'Used in the automaton of %c.',
    [nvalue]).

ctr_arguments(
    among_diff_0,
    ['NVAR'-dvar,'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    among_diff_0,
    ['NVAR'>=0,
     'NVAR'=<size('VARIABLES'),
     required('VARIABLES',var)]).

ctr_graph(
    among_diff_0,
    ['VARIABLES'],
    1,
    ['SELF'>>collection(variables)],
    [variables^var=\=0],
    ['NARC'='NVAR']).

ctr_example(
    among_diff_0,
    among_diff_0(3,[var-0],[var-5],[var-5],[var-0],[var-1])).

among_diff_0(A,B) :-
    among_diff_0_signature(B,C),
    automaton(
        C,
        D,
        C,
        0..1,
        [source(s),sink(t)],
        [arc(s,0,s),arc(s,1,s,[E+1]),arc(s,$,t)],
        [E],
        [0],
        [A]).

```

```
among_diff_0_signature([], []).
```

```
among_diff_0_signature([[var-A]|B], [C|D]) :-  
    A#\=0#<=>C,  
    among_diff_0_signature(B,D).
```

B.14 among_interval

```

ctr_automaton(among_interval,among_interval).

ctr_date(among_interval,['20030820','20040530']).

ctr_origin(among_interval,'Derived from %c.',[among]).

ctr_arguments(
    among_interval,
    ['NVAR'-dvar,
     'VARIABLES'-collection(var-dvar),
     'LOW'-int,
     'UP'-int]).

ctr_restrictions(
    among_interval,
    ['NVAR'>=0,
     'NVAR'=<size('VARIABLES'),
     required('VARIABLES',var),
     'LOW'=<'UP'] ).

ctr_graph(
    among_interval,
    ['VARIABLES'],
    1,
    ['SELF'>>collection(variables)],
    ['LOW'=<variables^var,variables^var=<'UP'],
    ['NARC'='NVAR'] ).

ctr_example(
    among_interval,
    among_interval(
        3,
        [[var-4],[var-5],[var-8],[var-4],[var-1]],
        3,
        5)).

among_interval(A,B,C,D) :-
    among_interval_signature(B,E,C,D),
    automaton(
        E,
        F,
        E,
        0..1,
        [source(s),sink(t)],

```



```
[arc(s,0,s),arc(s,1,s,[G+1]),arc(s,$,t)],  
[G],  
[0],  
[A]).
```

```
among_interval_signature([],[],A,B).
```

```
among_interval_signature([[var-A]|B],[C|D],E,F) :-  
E#=<A#/\A#=<F#<=>C,  
among_interval_signature(B,D,E,F).
```

B.15 among_low_up

```

ctr_automaton(among_low_up, among_low_up) .

ctr_date(among_low_up, ['20030820', '20040530']) .

ctr_origin(among_low_up, '\\cite{BeldiceanuContejean94}', []).

ctr_arguments(
    among_low_up,
    ['LOW'-int,
     'UP'-int,
     'VARIABLES'-collection(var-dvar),
     'VALUES'-collection(val-int)]).

ctr_restrictions(
    among_low_up,
    ['LOW'>=0,
     'LOW'=<size('VARIABLES'),
     'UP'>='LOW',
     required('VARIABLES', var),
     required('VALUES', val),
     distinct('VALUES', val)]).

ctr_graph(
    among_low_up,
    ['VARIABLES', 'VALUES'],
    2,
    ['PRODUCT'>>collection(variables, values)],
    [variables^var=values^val],
    ['NARC'>='LOW', 'NARC'=<'UP'] ).

ctr_example(
    among_low_up,
    among_low_up(
        1,
        2,
        [[var-9], [var-2], [var-4], [var-5]],
        [[val-0], [val-2], [val-4], [val-6], [val-8]])).

among_low_up(A, B, C, D) :-
    col_to_list(D, E),
    list_to_fdset(E, F),
    among_low_up_signature(C, G, F),
    in(H, A..B),
    automaton(

```

```
G,  
I,  
G,  
0..1,  
[source(s),sink(t)],  
[arc(s,0,s),arc(s,1,s,[J+1]),arc(s,$,t)],  
[J],  
[0],  
[H]).
```

```
among_low_up_signature([],[],A).
```

```
among_low_up_signature([[var-A]|B],[C|D],E) :-  
    in_set(A,E) #<=>C,  
    among_low_up_signature(B,D,E).
```

B.16 among_modulo

```

ctr_automaton(among_modulo,among_modulo).

ctr_date(among_modulo,['20030820','20040530']).

ctr_origin(among_modulo,'Derived from %c.',[among]).

ctr_arguments(
    among_modulo,
    ['NVAR'-dvar,
     'VARIABLES'-collection(var-dvar),
     'REMAINDER'-int,
     'QUOTIENT'-int]).

ctr_restrictions(
    among_modulo,
    ['NVAR'>=0,
     'NVAR'=<size('VARIABLES'),
     required('VARIABLES',var),
     'REMAINDER'>=0,
     'REMAINDER'<'QUOTIENT',
     'QUOTIENT'>0]).

ctr_graph(
    among_modulo,
    ['VARIABLES'],
    1,
    ['SELF'>>collection(variables)],
    [variables^var mod 'QUOTIENT'='REMAINDER'],
    ['NARC'='NVAR']).

ctr_example(
    among_modulo,
    among_modulo(
        3,
        [[var-4],[var-5],[var-8],[var-4],[var-1]],
        0,
        2)).

among_modulo(A,B,C,D) :-
    among_modulo_signature(B,E,C,D),
    automaton(
        E,
        F,
        E,

```

```
0..1,  
[source(s), sink(t)],  
[arc(s, 0, s), arc(s, 1, s, [G+1]), arc(s, $, t)],  
[G],  
[0],  
[A]).
```

```
among_modulo_signature([], [], A, B).
```

```
among_modulo_signature([[var-A] | B], [C | D], E, F) :-  
    A mod F #= E #<=> C,  
    among_modulo_signature(B, D, E, F).
```

B.17 among_seq

```

ctr_date(among_seq, ['20000128', '20030820']).

ctr_origin(among_seq, '\\cite{BeldiceanuContejean94}', []).

ctr_arguments(
  among_seq,
  ['LOW'-int,
   'UP'-int,
   'SEQ'-int,
   'VARIABLES'-collection(var-dvar),
   'VALUES'-collection(val-int)].

ctr_restrictions(
  among_seq,
  ['LOW'>=0,
   'LOW'=<size('VARIABLES'),
   'UP'>='LOW',
   'SEQ'>0,
   'SEQ'>='LOW',
   'SEQ'=<size('VARIABLES'),
   required('VARIABLES', var),
   required('VALUES', val),
   distinct('VALUES', val)].

ctr_graph(
  among_seq,
  ['VARIABLES'],
  'SEQ',
  ['PATH'>>collection],
  [among_low_up('LOW', 'UP', collection, 'VALUES')],
  ['NARC'=size('VARIABLES')-'SEQ'+1]).

ctr_example(
  among_seq,
  among_seq(
    1,
    2,
    4,
    [[var-9],
     [var-2],
     [var-4],
     [var-5],
     [var-5],
     [var-7],

```

```
[var-2]],  
[[val-0],[val-2],[val-4],[val-6],[val-8]])).
```

B.18 arith

```

ctr_automaton(arith,arith).

ctr_date(arith,['20040814']).

ctr_origin(
    arith,
    'Used in the definition of several automata',
    []).

ctr_arguments(
    arith,
    ['VARIABLES'-collection(var-dvar),
     'RELOP'-atom,
     'VALUE'-int]).

ctr_restrictions(
    arith,
    [required('VARIABLES',var),
     in_list('RELOP',[=,\=,<,>=,>,=<])]).

ctr_graph(
    arith,
    ['VARIABLES'],
    1,
    ['SELF'>>collection(variables)],
    ['RELOP'(variables^var,'VALUE')],
    ['NARC'=size('VARIABLES')]).

ctr_example(
    arith,
    arith([[var-4],[var-5],[var-7],[var-4],[var-5]],<,9)).

arith(A,B,C) :-
    arith_signature(A,D,B,C),
    automaton(
        D,
        E,
        D,
        0..1,
        [source(s),sink(t)],
        [arc(s,1,s),arc(s,$,t)],
        [],
        [],
        []).

```



```
arith_signature([], [], A, B).
```

```
arith_signature([[var-A] | B], [C|D], =, E) :-
    A#E#<=>C,
    arith_signature(B, D, =, E).
```

```
arith_signature([[var-A] | B], [C|D], =\=, E) :-
    A#\=E#<=>C,
    arith_signature(B, D, =\=, E).
```

```
arith_signature([[var-A] | B], [C|D], <, E) :-
    A#<E#<=>C,
    arith_signature(B, D, <, E).
```

```
arith_signature([[var-A] | B], [C|D], >=, E) :-
    A#>=E#<=>C,
    arith_signature(B, D, >=, E).
```

```
arith_signature([[var-A] | B], [C|D], >, E) :-
    A#>E#<=>C,
    arith_signature(B, D, >, E).
```

```
arith_signature([[var-A] | B], [C|D], =<, E) :-
    A#=<E#<=>C,
    arith_signature(B, D, =<, E).
```

B.19 arith_or

```

ctr_automaton(arith_or, arith_or).

ctr_date(arith_or, ['20040814']).

ctr_origin(
    arith_or,
    'Used in the definition of several automata',
    []).

ctr_arguments(
    arith_or,
    ['VARIABLES1'-collection(var-dvar),
     'VARIABLES2'-collection(var-dvar),
     'RELOP'-atom,
     'VALUE'-int]).

ctr_restrictions(
    arith_or,
    [required('VARIABLES1', var),
     required('VARIABLES2', var),
     size('VARIABLES1')=size('VARIABLES2'),
     in_list('RELOP', [=,=\=, <, >=, >, =<])]).

ctr_graph(
    arith_or,
    ['VARIABLES1', 'VARIABLES2'],
    2,
    ['PRODUCT' (=)>>collection(variables1, variables2)],
    [#\(/('RELOP' (variables1^var, 'VALUE'),
         'RELOP' (variables2^var, 'VALUE')))],
    ['NARC'=size('VARIABLES1')]).

ctr_example(
    arith_or,
    arith_or(
        [[var-0], [var-1], [var-0], [var-0], [var-1]],
        [[var-0], [var-0], [var-0], [var-1], [var-0]],
        =,
        0)).

arith_or(A, B, C, D) :-
    arith_or_signature(A, B, E, C, D),
    automaton(
        E,

```

```

F,
E,
0..1,
[source(s), sink(t)],
[arc(s, l, s), arc(s, $, t)],
[],
[],
[]).

```

```
arith_or_signature([], [], [], A, B).
```

```
arith_or_signature([[var-A] | B], [[var-C] | D], [E | F], =, G) :-
  A#=#G#\C#=#G#<=>E,
  arith_or_signature(B, D, F, =, G).
```

```
arith_or_signature([[var-A] | B], [[var-C] | D], [E | F], =\=, G) :-
  A#\=#G#\C#\=#G#<=>E,
  arith_or_signature(B, D, F, =\=, G).
```

```
arith_or_signature([[var-A] | B], [[var-C] | D], [E | F], <, G) :-
  A#<G#\C#<G#<=>E,
  arith_or_signature(B, D, F, <, G).
```

```
arith_or_signature([[var-A] | B], [[var-C] | D], [E | F], >=, G) :-
  A#>=G#\C#>=G#<=>E,
  arith_or_signature(B, D, F, >=, G).
```

```
arith_or_signature([[var-A] | B], [[var-C] | D], [E | F], >, G) :-
  A#>G#\C#>G#<=>E,
  arith_or_signature(B, D, F, >, G).
```

```
arith_or_signature([[var-A] | B], [[var-C] | D], [E | F], =<, G) :-
  A#=#<G#\C#=#<G#<=>E,
  arith_or_signature(B, D, F, =<, G).
```

B.20 arith_sliding

```

ctr_automaton(arith_sliding, arith_sliding).

ctr_date(arith_sliding, ['20040814']).

ctr_origin(
    arith_sliding,
    'Used in the definition of some automaton',
    []).

ctr_arguments(
    arith_sliding,
    ['VARIABLES'-collection(var-dvar),
     'RELOP'-atom,
     'VALUE'-int]).

ctr_restrictions(
    arith_sliding,
    [required('VARIABLES', var),
     in_list('RELOP', [=,=\=, <, >=, >, =<])]).

ctr_graph(
    arith_sliding,
    ['VARIABLES'],
    *,
    ['PATH_1'>>collection],
    [arith(collection, 'RELOP', 'VALUE')],
    ['NARC'=size('VARIABLES')]).

ctr_example(
    arith_sliding,
    arith_sliding(
        [[var-0],
         [var-0],
         [var-1],
         [var-2],
         [var-0],
         [var-0],
         [var- -3]],
        <,
        4)).

arith_sliding(A, =, B) :-
    length(A, C),
    length(D, C),

```

```

domain(D,0,0),
arith_sliding_signature(A,E,D),
automaton(
  E,
  F,
  D,
  0..0,
  [source(s),node(i),sink(t)],
  [arc(s,0,i,[G+F]),
   arc(s,$,t,[G]),
   arc(i,0,i,(G#=B->[G+F])),
   arc(i,$,t,(G#=B->[G]))],
  [G],
  [0],
  [H]).

```

```

arith_sliding(A,=\=,B) :-
  length(A,C),
  length(D,C),
  domain(D,0,0),
  arith_sliding_signature(A,E,D),
  automaton(
    E,
    F,
    D,
    0..0,
    [source(s),node(i),sink(t)],
    [arc(s,0,i,[G+F]),
     arc(s,$,t,[G]),
     arc(i,0,i,(G#\=B->[G+F])),
     arc(i,$,t,(G#\=B->[G]))],
    [G],
    [0],
    [H]).

```

```

arith_sliding(A,<,B) :-
  length(A,C),
  length(D,C),
  domain(D,0,0),
  arith_sliding_signature(A,E,D),
  automaton(
    E,
    F,
    D,
    0..0,
    [source(s),node(i),sink(t)],

```

```

[arc(s,0,i,[G+F]),
 arc(s,$,t,[G]),
 arc(i,0,i,(G#<B->[G+F])),
 arc(i,$,t,(G#<B->[G]))],
[G],
[0],
[H]).

```

```

arith_sliding(A,>=,B) :-
  length(A,C),
  length(D,C),
  domain(D,0,0),
  arith_sliding_signature(A,E,D),
  automaton(
    E,
    F,
    D,
    0..0,
    [source(s),node(i),sink(t)],
    [arc(s,0,i,[G+F]),
     arc(s,$,t,[G]),
     arc(i,0,i,(G#>=B->[G+F])),
     arc(i,$,t,(G#>=B->[G]))],
    [G],
    [0],
    [H]).

```

```

arith_sliding(A,>,B) :-
  length(A,C),
  length(D,C),
  domain(D,0,0),
  arith_sliding_signature(A,E,D),
  automaton(
    E,
    F,
    D,
    0..0,
    [source(s),node(i),sink(t)],
    [arc(s,0,i,[G+F]),
     arc(s,$,t,[G]),
     arc(i,0,i,(G#>B->[G+F])),
     arc(i,$,t,(G#>B->[G]))],
    [G],
    [0],
    [H]).

```

```

arith_sliding(A,=<,B) :-
    length(A,C),
    length(D,C),
    domain(D,0,0),
    arith_sliding_signature(A,E,D),
    automaton(
        E,
        F,
        D,
        0..0,
        [source(s),node(i),sink(t)],
        [arc(s,0,i,[G+F]),
         arc(s,$,t,[G]),
         arc(i,0,i,(G#=<B->[G+F])),
         arc(i,$,t,(G#=<B->[G]))],
        [G],
        [0],
        [H]).

arith_sliding_signature([],[],[]).

arith_sliding_signature([[var-A]|B],[A|C],[0|D]) :-
    arith_sliding_signature(B,C,D).

```

B.21 assign_and_counts

```

ctr_date(assign_and_counts, ['20000128', '20030820']).

ctr_origin(assign_and_counts, 'N.~Beldiceanu', []).

ctr_arguments(
  assign_and_counts,
  ['COLOURS'-collection(val-int),
   'ITEMS'-collection(bin-dvar, colour-dvar),
   'RELOP'-atom,
   'LIMIT'-dvar]).

ctr_restrictions(
  assign_and_counts,
  [required('COLOURS', val),
   distinct('COLOURS', val),
   required('ITEMS', [bin, colour]),
   in_list('RELOP', [=, =\=, <, >=, >, =<])]).

ctr_derived_collections(
  assign_and_counts,
  [col('VALUES'-collection(val-int),
      [item(val-'COLOURS'^val)])]).

ctr_graph(
  assign_and_counts,
  ['ITEMS', 'ITEMS'],
  2,
  ['PRODUCT'>>collection(items1, items2)],
  [items1^bin=items2^bin],
  [],
  [>>('SUCC',
      [source,
       -(variables,
          col('VARIABLES'-collection(var-dvar),
              [item(var-'ITEMS'^colour)])])]),
   [counts('VALUES', variables, 'RELOP', 'LIMIT')]).

ctr_example(
  assign_and_counts,
  assign_and_counts(
    [[val-4]],
    [[bin-1, colour-4],
     [bin-3, colour-4],
     [bin-1, colour-4],

```



```
[bin-1,colour-5]],  
=<,  
2)).
```

B.22 assign_and_nvalues

```
ctr_date(
  assign_and_nvalues,
  ['20000128', '20030820', '20040530', '20050321']).
```

```
ctr_origin(
  assign_and_nvalues,
  'Derived from %c and %c.',
  [assign_and_counts, nvalues]).
```

```
ctr_arguments(
  assign_and_nvalues,
  ['ITEMS'-collection(bin-dvar, value-dvar),
   'RELOP'-atom,
   'LIMIT'-dvar]).
```

```
ctr_restrictions(
  assign_and_nvalues,
  [required('ITEMS', [bin, value]),
   in_list('RELOP', [=, =\=, <, >=, >, =<])]).
```

```
ctr_graph(
  assign_and_nvalues,
  ['ITEMS', 'ITEMS'],
  2,
  ['PRODUCT'>>collection(items1, items2)],
  [items1^bin=items2^bin],
  [],
  [>>('SUCC',
       [source,
        -(variables,
          col('VARIABLES'-collection(var-dvar),
            [item(var-'ITEMS'^value)])])]),
   [nvalues(variables, 'RELOP', 'LIMIT')]).
```

```
ctr_example(
  assign_and_nvalues,
  assign_and_nvalues(
    [[bin-2, value-3],
     [bin-1, value-5],
     [bin-2, value-3],
     [bin-2, value-3],
     [bin-2, value-4]],
    =<,
    2)).
```


B.23 atleast

```

ctr_automaton(atleast, atleast) .

ctr_date(atleast, ['20030820', '20040807']) .

ctr_origin(atleast, 'CHIP', []).

ctr_arguments(
    atleast,
    ['N'-int, 'VARIABLES'-collection(var-dvar), 'VALUE'-int]) .

ctr_restrictions(
    atleast,
    ['N'>=0, 'N'=<size('VARIABLES'), required('VARIABLES', var)]) .

ctr_graph(
    atleast,
    ['VARIABLES'],
    1,
    ['SELF'>>collection(variables)],
    [variables^var='VALUE'],
    ['NARC'>='N']) .

ctr_example(
    atleast,
    atleast(2, [[var-4], [var-2], [var-4], [var-5]], 4)) .

atleast(A,B,C) :-
    atleast_signature(B,D,C),
    length(B,E),
    in(F,A..E),
    automaton(
        D,
        G,
        D,
        0..1,
        [source(s), sink(t)],
        [arc(s,0,s), arc(s,1,s, [H+1]), arc(s,$,t)],
        [H],
        [0],
        [F]) .

atleast_signature([], [], A) .

atleast_signature([[var-A]|B], [C|D], E) :-

```

$A \# = E \# \Leftrightarrow C$,
atleast_signature(B,D,E) .

B.24 atmost

```

ctr_automaton(atmost,atmost).

ctr_date(atmost,['20030820','20040807']).

ctr_origin(atmost,'CHIP',[ ]).

ctr_arguments(
    atmost,
    ['N'-int,'VARIABLES'-collection(var-dvar),'VALUE'-int]).

ctr_restrictions(atmost,['N'>=0,required('VARIABLES',var)]).

ctr_graph(
    atmost,
    ['VARIABLES'],
    1,
    ['SELF'>>collection(variables)],
    [variables^var='VALUE'],
    ['NARC'=<'N']).

ctr_example(
    atmost,
    atmost(1,[[var-4],[var-2],[var-4],[var-5]],2)).

atmost(A,B,C) :-
    atmost_signature(B,D,C),
    in(E,0..A),
    automaton(
        D,
        F,
        D,
        0..1,
        [source(s),sink(t)],
        [arc(s,0,s),arc(s,1,s,[G+1]),arc(s,$,t)],
        [G],
        [0],
        [E]).

atmost_signature([],[],A).

atmost_signature([[var-A]|B],[C|D],E) :-
    A#=E#<=>C,
    atmost_signature(B,D,E).

```

B.25 balance

```
ctr_date(balance, ['20000128', '20030820']).

ctr_origin(balance, 'N.~Beldiceanu', []).

ctr_arguments(
    balance,
    ['BALANCE'-dvar, 'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    balance,
    ['BALANCE'>=0,
     'BALANCE'=<size('VARIABLES'),
     required('VARIABLES', var)]).

ctr_graph(
    balance,
    ['VARIABLES'],
    2,
    ['CLIQUE'>>collection(variables1, variables2)],
    [variables1^var=variables2^var],
    ['RANGE_NSCC'='BALANCE']).

ctr_example(
    balance,
    balance(2, [[var-3], [var-1], [var-7], [var-1], [var-1]])).
```

B.26 balance_interval

```
ctr_date(balance_interval, ['20030820']).

ctr_origin(balance_interval, 'Derived from %c.', [balance]).

ctr_arguments(
    balance_interval,
    ['BALANCE'-dvar,
     'VARIABLES'-collection(var-dvar),
     'SIZE_INTERVAL'-int]).

ctr_restrictions(
    balance_interval,
    ['BALANCE'>=0,
     'BALANCE'=<size('VARIABLES'),
     required('VARIABLES', var),
     'SIZE_INTERVAL'>0]).

ctr_graph(
    balance_interval,
    ['VARIABLES'],
    2,
    ['CLIQUE'>>collection(variables1, variables2)],
    [(variables1^var/'SIZE_INTERVAL',
     variables2^var/'SIZE_INTERVAL')],
    ['RANGE_NSCC'='BALANCE']).

ctr_example(
    balance_interval,
    balance_interval(
        3,
        [[var-6], [var-4], [var-3], [var-3], [var-4]],
        3)).
```


B.27 balance_modulo

```
ctr_date(balance_modulo, ['20030820']).

ctr_origin(balance_modulo, 'Derived from %c.', [balance]).

ctr_arguments(
    balance_modulo,
    ['BALANCE'-dvar, 'VARIABLES'-collection(var-dvar), 'M'-int]).

ctr_restrictions(
    balance_modulo,
    ['BALANCE'>=0,
     'BALANCE'=<size('VARIABLES'),
     required('VARIABLES', var),
     'M'>0]).

ctr_graph(
    balance_modulo,
    ['VARIABLES'],
    2,
    ['CLIQUE'>>collection(variables1, variables2)],
    [variables1^var mod 'M'=variables2^var mod 'M'],
    ['RANGE_NSCC'='BALANCE']).

ctr_example(
    balance_modulo,
    balance_modulo(
        2,
        [[var-6], [var-1], [var-7], [var-1], [var-5]],
        3)).
```

B.28 balance_partition

```

ctr_date(balance_partition, ['20030820']).

ctr_origin(balance_partition, 'Derived from %c.', [balance]).

ctr_types(balance_partition, ['VALUES'-collection(val-int)]).

ctr_arguments(
    balance_partition,
    ['BALANCE'-dvar,
     'VARIABLES'-collection(var-dvar),
     'PARTITIONS'-collection(p-'VALUES')]).

ctr_restrictions(
    balance_partition,
    [required('VALUES', val),
     distinct('VALUES', val),
     'BALANCE'>=0,
     'BALANCE'=<size('VARIABLES'),
     required('VARIABLES', var),
     required('PARTITIONS', p),
     size('PARTITIONS')>=2]).

ctr_graph(
    balance_partition,
    ['VARIABLES'],
    2,
    ['CLIQUE'>>collection(variables1, variables2)],
    [in_same_partition(
        variables1^var,
        variables2^var,
        'PARTITIONS')],
    ['RANGE_NSCC'='BALANCE']).

ctr_example(
    balance_partition,
    balance_partition(
        1,
        [[var-6], [var-2], [var-6], [var-4], [var-4]],
        [[p-[[val-1], [val-3]]],
         [p-[[val-4]]],
         [p-[[val-2], [val-6]]]])).

```

B.29 bin_packing

```

ctr_date(bin_packing, ['20000128', '20030820', '20040530']).

ctr_origin(bin_packing, 'Derived from %c.', [cumulative]).

ctr_arguments(
    bin_packing,
    ['CAPACITY'-int, 'ITEMS'-collection(bin-dvar, weight-int)]).

ctr_restrictions(
    bin_packing,
    ['CAPACITY'>=0,
     required('ITEMS', [bin, weight]),
     'ITEMS'^weight>=0,
     'ITEMS'^weight=<'CAPACITY'] ).

ctr_graph(
    bin_packing,
    ['ITEMS', 'ITEMS'],
    2,
    ['PRODUCT'>>collection(items1, items2)],
    [items1^bin=items2^bin],
    [],
    [>>('SUCC',
         [source,
          -(variables,
             col('VARIABLES'-collection(var-dvar),
                 [item(var-'ITEMS'^weight)]))]]],
    [sum_ctr(variables, =<, 'CAPACITY')]).

ctr_example(
    bin_packing,
    bin_packing(
        5,
        [[bin-3, weight-4], [bin-1, weight-3], [bin-3, weight-1]]).

```

B.30 binary_tree

```

ctr_date(binary_tree, ['20000128', '20030820']).

ctr_origin(binary_tree, 'Derived from %c.', [tree]).

ctr_arguments(
    binary_tree,
    ['NTREES'-dvar, 'NODES'-collection(index-int, succ-dvar)]).

ctr_restrictions(
    binary_tree,
    ['NTREES'>=0,
     required('NODES', [index, succ]),
     'NODES'^index>=1,
     'NODES'^index<=size('NODES'),
     distinct('NODES', index),
     'NODES'^succ>=1,
     'NODES'^succ<=size('NODES')]).

ctr_graph(
    binary_tree,
    ['NODES'],
    2,
    ['CLIQUE'>>collection(nodes1, nodes2)],
    [nodes1^succ=nodes2^index],
    ['MAX_NSCC'=<1, 'NCC'='NTREES', 'MAX_ID'=<2]).

ctr_example(
    binary_tree,
    binary_tree(
        2,
        [[index-1, succ-1],
         [index-2, succ-3],
         [index-3, succ-5],
         [index-4, succ-7],
         [index-5, succ-1],
         [index-6, succ-1],
         [index-7, succ-7],
         [index-8, succ-5]])).

```

B.31 cardinality_atleast

```
ctr_date(cardinality_atleast, ['20030820', '20040530']).
```

```
ctr_origin(
    cardinality_atleast,
    'Derived from %c.',
    [global_cardinality]).
```

```
ctr_arguments(
    cardinality_atleast,
    ['ATLEAST'-dvar,
     'VARIABLES'-collection(var-dvar),
     'VALUES'-collection(val-int)].
```

```
ctr_restrictions(
    cardinality_atleast,
    ['ATLEAST'>=0,
     'ATLEAST'=<size('VARIABLES'),
     required('VARIABLES', var),
     required('VALUES', val),
     distinct('VALUES', val)].
```

```
ctr_graph(
    cardinality_atleast,
    ['VARIABLES', 'VALUES'],
    2,
    ['PRODUCT'>>collection(variables, values)],
    [variables^var=\=values^val],
    ['MAX_ID'=size('VARIABLES')-'ATLEAST']).
```

```
ctr_example(
    cardinality_atleast,
    cardinality_atleast(
        1,
        [[var-3], [var-3], [var-8]],
        [[val-3], [val-8]])).
```

B.32 cardinality_atmost

```

ctr_date(cardinality_atmost, ['20030820', '20040530']).

ctr_origin(
    cardinality_atmost,
    'Derived from %c.',
    [global_cardinality]).

ctr_arguments(
    cardinality_atmost,
    ['ATMOST'-dvar,
     'VARIABLES'-collection(var-dvar),
     'VALUES'-collection(val-int)]).

ctr_restrictions(
    cardinality_atmost,
    ['ATMOST'>=0,
     'ATMOST'=<size('VARIABLES'),
     required('VARIABLES', var),
     required('VALUES', val),
     distinct('VALUES', val)]).

ctr_graph(
    cardinality_atmost,
    ['VARIABLES', 'VALUES'],
    2,
    ['PRODUCT'>>collection(variables, values)],
    [variables^var=values^val],
    ['MAX_ID'='ATMOST']).

ctr_example(
    cardinality_atmost,
    cardinality_atmost(
        2,
        [[var-2], [var-1], [var-7], [var-1], [var-2]],
        [[val-5], [val-7], [val-2], [val-9]])).

```

B.33 cardinality_atmost_partition

```

ctr_date(cardinality_atmost_partition, ['20030820']).

ctr_origin(
    cardinality_atmost_partition,
    'Derived from %c.',
    [global_cardinality]).

ctr_types(
    cardinality_atmost_partition,
    ['VALUES'-collection(val-int)]).

ctr_arguments(
    cardinality_atmost_partition,
    ['ATMOST'-dvar,
     'VARIABLES'-collection(var-dvar),
     'PARTITIONS'-collection(p-'VALUES')]).

ctr_restrictions(
    cardinality_atmost_partition,
    [required('VALUES', val),
     distinct('VALUES', val),
     'ATMOST' >= 0,
     'ATMOST' <= size('VARIABLES'),
     required('VARIABLES', var),
     required('PARTITIONS', p),
     size('PARTITIONS') >= 2]).

ctr_graph(
    cardinality_atmost_partition,
    ['VARIABLES', 'PARTITIONS'],
    2,
    ['PRODUCT' >> collection(variables, partitions)],
    [in(variables^var, partitions^p)],
    ['MAX_ID' = 'ATMOST']).

ctr_example(
    cardinality_atmost_partition,
    cardinality_atmost_partition(
        2,
        [[var-2], [var-3], [var-7], [var-1], [var-6], [var-0]],
        [[p-[val-1], [val-3]]],
        [p-[val-4]],
        [p-[val-2], [val-6]]])).

```

B.34 change

```

ctr_automaton(change, change) .

ctr_date(change, ['20000128', '20030820', '20040530']) .

ctr_origin(change, 'CHIP', []).

ctr_synonyms(change, [nbchanges, similarity]) .

ctr_arguments(
    change,
    ['NCHANGE'-dvar,
     'VARIABLES'-collection(var-dvar),
     'CTR'-atom]) .

ctr_restrictions(
    change,
    ['NCHANGE'>=0,
     'NCHANGE'<size('VARIABLES'),
     required('VARIABLES', var),
     in_list('CTR', [=,=\=, <, >=, >, =<])]) .

ctr_graph(
    change,
    ['VARIABLES'],
    2,
    ['PATH'>>collection(variables1, variables2)],
    ['CTR'(variables1^var, variables2^var)],
    ['NARC'='NCHANGE']) .

ctr_example(
    change,
    [change(3, [[var-4], [var-4], [var-3], [var-4], [var-1]], =\=),
     change(1, [[var-1], [var-2], [var-4], [var-3], [var-7]], >)]) .

change(A, B, C) :-
    change_signature(B, D, C),
    automaton(
        D,
        E,
        D,
        0..1,
        [source(s), sink(t)],
        [arc(s, 0, s), arc(s, 1, s, [F+1]), arc(s, $, t)],
        [F],

```



```
[0],
[A]).
```

```
change_signature([], [], A).
```

```
change_signature([A], [], B) :-
    !.
```

```
change_signature([[var-A], [var-B] | C], [D|E], =) :-
    !,
    A#B#<=>D,
    change_signature([ [var-B] | C], E, =).
```

```
change_signature([[var-A], [var-B] | C], [D|E], =\=) :-
    !,
    A#\=B#<=>D,
    change_signature([ [var-B] | C], E, =\=).
```

```
change_signature([[var-A], [var-B] | C], [D|E], <) :-
    !,
    A#<B#<=>D,
    change_signature([ [var-B] | C], E, <).
```

```
change_signature([[var-A], [var-B] | C], [D|E], >=) :-
    !,
    A#>=B#<=>D,
    change_signature([ [var-B] | C], E, >=).
```

```
change_signature([[var-A], [var-B] | C], [D|E], >) :-
    !,
    A#>B#<=>D,
    change_signature([ [var-B] | C], E, >).
```

```
change_signature([[var-A], [var-B] | C], [D|E], =<) :-
    !,
    A#=<B#<=>D,
    change_signature([ [var-B] | C], E, =<).
```

B.35 change_continuity

```

ctr_automaton(change_continuity,change_continuity).

ctr_date(change_continuity,['20000128','20030820','20040530']).

ctr_origin(change_continuity,'N.~Beldiceanu',[]).

ctr_arguments(
  change_continuity,
  ['NB_PERIOD_CHANGE'-dvar,
   'NB_PERIOD_CONTINUITY'-dvar,
   'MIN_SIZE_CHANGE'-dvar,
   'MAX_SIZE_CHANGE'-dvar,
   'MIN_SIZE_CONTINUITY'-dvar,
   'MAX_SIZE_CONTINUITY'-dvar,
   'NB_CHANGE'-dvar,
   'NB_CONTINUITY'-dvar,
   'VARIABLES'-collection(var-dvar),
   'CTR'-atom]).

ctr_restrictions(
  change_continuity,
  ['NB_PERIOD_CHANGE'>=0,
   'NB_PERIOD_CONTINUITY'>=0,
   'MIN_SIZE_CHANGE'>=0,
   'MAX_SIZE_CHANGE'>='MIN_SIZE_CHANGE',
   'MIN_SIZE_CONTINUITY'>=0,
   'MAX_SIZE_CONTINUITY'>='MIN_SIZE_CONTINUITY',
   'NB_CHANGE'>=0,
   'NB_CONTINUITY'>=0,
   required('VARIABLES',var),
   in_list('CTR',[=,\=,<,>=>,>=<])]).

ctr_graph(
  change_continuity,
  ['VARIABLES'],
  2,
  ['PATH'>>collection(variables1,variables2)],
  ['CTR'(variables1^var,variables2^var)],
  ['NCC'='NB_PERIOD_CHANGE',
   'MIN_NCC'='MIN_SIZE_CHANGE',
   'MAX_NCC'='MAX_SIZE_CHANGE',
   'NARC'='NB_CHANGE']]).

ctr_graph(

```

```

change_continuity,
['VARIABLES'],
2,
['PATH'>>collection(variables1,variables2)],
[#\'CTR' (variables1^var,variables2^var)],
['NCC'='NB_PERIOD_CONTINUITY',
 'MIN_NCC'='MIN_SIZE_CONTINUITY',
 'MAX_NCC'='MAX_SIZE_CONTINUITY',
 'NARC'='NB_CONTINUITY'] ).

ctr_example(
change_continuity,
change_continuity(
  3,
  2,
  2,
  4,
  2,
  4,
  6,
  4,
  [[var-1],
   [var-3],
   [var-1],
   [var-8],
   [var-8],
   [var-4],
   [var-7],
   [var-7],
   [var-7],
   [var-7],
   [var-2]],
  =\=) ).

change_continuity(A,B,C,D,E,F,G,H,I,J) :-
length(I,K),
change_continuity_signature(I,L,1,J),
change_continuity_signature(I,M,0,J),
change_continuity_nb_period(A,L),
change_continuity_nb_period(B,M),
change_continuity_min_size(C,L),
change_continuity_min_size(E,M),
change_continuity_max_size(D,L),
change_continuity_max_size(F,M),
change_continuity_nb(G,L),
change_continuity_nb(H,M) .

```

```

change_continuity_nb_period(A,B) :-
    automaton(
        B,
        C,
        B,
        0..1,
        [source(s),node(i),sink(t)],
        [arc(s,0,s),
         arc(s,1,i,[D+1]),
         arc(s,$,t),
         arc(i,1,i),
         arc(i,0,s),
         arc(i,$,t)],
        [D],
        [0],
        [A]).

```

```

change_continuity_min_size(A,B) :-
    automaton(
        B,
        C,
        B,
        0..1,
        [source(s),node(i),node(j),node(k),sink(t)],
        [arc(s,0,s),
         arc(s,1,i,[D,2]),
         arc(s,$,t,[D,E]),
         arc(i,0,j,[E,E]),
         arc(i,1,i,[D,E+1]),
         arc(i,$,t,[E,E]),
         arc(j,0,j),
         arc(j,1,k,[D,2]),
         arc(j,$,t,[D,E]),
         arc(k,0,j,[min(D,E),E]),
         arc(k,1,k,[D,E+1]),
         arc(k,$,t,[min(D,E),E])],
        [D,E],
        [0,1],
        [A,F]).

```

```

change_continuity_max_size(A,B) :-
    automaton(
        B,
        C,
        B,

```

```

0..1,
[source(s),node(i),sink(t)],
[arc(s,0,s,[D,E]),
 arc(s,1,i,[D,E+1]),
 arc(s,$,t,[D,E]),
 arc(i,0,i,[max(D,E),1]),
 arc(i,1,i,[D,E+1]),
 arc(i,$,t,[max(D,E),E])],
[D,E],
[0,1],
[A,F]).

```

```

change_continuity_nb(A,B) :-
  automaton(
    B,
    C,
    B,
    0..1,
    [source(s),sink(t)],
    [arc(s,0,s),arc(s,1,s,[D+1]),arc(s,$,t)],
    [D],
    [0],
    [A]).

```

```

change_continuity_signature([],[],A,B).

```

```

change_continuity_signature([A],[],B,C) :-
  !.

```

```

change_continuity_signature([[var-A],[var-B]|C],[D|E],1,=) :-
  !,
  A#=B#<=>D,
  change_continuity_signature([[var-B]|C],E,1,=).

```

```

change_continuity_signature([[var-A],[var-B]|C],[D|E],1,=\=) :-
  !,
  A#\=B#<=>D,
  change_continuity_signature([[var-B]|C],E,1,=\=).

```

```

change_continuity_signature([[var-A],[var-B]|C],[D|E],1,<) :-
  !,
  A#<B#<=>D,
  change_continuity_signature([[var-B]|C],E,1,<).

```

```

change_continuity_signature([[var-A],[var-B]|C],[D|E],1,>=) :-
  !,

```

```

A#>=B#<=>D,
change_continuity_signature([ [var-B] | C], E, 1, >=) .

change_continuity_signature([ [var-A], [var-B] | C], [D|E], 1, >) :-
!,
A#>B#<=>D,
change_continuity_signature([ [var-B] | C], E, 1, >) .

change_continuity_signature([ [var-A], [var-B] | C], [D|E], 1, =<) :-
!,
A#=<B#<=>D,
change_continuity_signature([ [var-B] | C], E, 1, =<) .

change_continuity_signature([ [var-A], [var-B] | C], [D|E], 0, =) :-
!,
A#\=B#<=>D,
change_continuity_signature([ [var-B] | C], E, 0, =) .

change_continuity_signature([ [var-A], [var-B] | C], [D|E], 0, =\=) :-
!,
A#=B#<=>D,
change_continuity_signature([ [var-B] | C], E, 0, =\=) .

change_continuity_signature([ [var-A], [var-B] | C], [D|E], 0, <) :-
!,
A#>=B#<=>D,
change_continuity_signature([ [var-B] | C], E, 0, <) .

change_continuity_signature([ [var-A], [var-B] | C], [D|E], 0, >=) :-
!,
A#<B#<=>D,
change_continuity_signature([ [var-B] | C], E, 0, >=) .

change_continuity_signature([ [var-A], [var-B] | C], [D|E], 0, >) :-
!,
A#=<B#<=>D,
change_continuity_signature([ [var-B] | C], E, 0, >) .

change_continuity_signature([ [var-A], [var-B] | C], [D|E], 0, =<) :-
!,
A#>B#<=>D,
change_continuity_signature([ [var-B] | C], E, 0, =<) .

```

B.36 change_pair

```

ctr_automaton(change_pair,change_pair).

ctr_date(change_pair,['20030820','20040530']).

ctr_origin(change_pair,'Derived from %c.',[change]).

ctr_arguments(
  change_pair,
  ['NCHANGE'-dvar,
   'PAIRS'-collection(x-dvar,y-dvar),
   'CTRX'-atom,
   'CTRY'-atom]).

ctr_restrictions(
  change_pair,
  ['NCHANGE'>=0,
   'NCHANGE'<size('PAIRS'),
   required('PAIRS',[x,y]),
   in_list('CTRX',[=,\=,<,>=>,>,<=]),
   in_list('CTRY',[=,\=,<,>=>,>,<=])]).

ctr_graph(
  change_pair,
  ['PAIRS'],
  2,
  ['PATH'>>collection(pairs1,pairs2)],
  ['CTRX'(pairs1^x,pairs2^x)#\/'CTRY'(pairs1^y,pairs2^y)],
  ['NARC'='NCHANGE']).

ctr_example(
  change_pair,
  change_pair(
    3,
    [[x-3,y-5],
     [x-3,y-7],
     [x-3,y-7],
     [x-3,y-8],
     [x-3,y-4],
     [x-3,y-7],
     [x-1,y-3],
     [x-1,y-6],
     [x-1,y-6],
     [x-3,y-7]],
    =\=,

```

>)).

```

change_pair(A,B,C,D) :-
    change_pair_signature(B,E,C,D),
    automaton(
        E,
        F,
        E,
        0..1,
        [source(s),sink(t)],
        [arc(s,0,s),arc(s,1,s,[G+1]),arc(s,$,t)],
        [G],
        [0],
        [A]).

change_pair_signature([],[],A,B).

change_pair_signature([A],[],B,C) :-
    !.

change_pair_signature([[x-A,y-B],[x-C,y-D]|E],[F|G],=,=) :-
    !,
    A#=C#\B#=D#<=>F,
    change_pair_signature([[x-C,y-D]|E],G,=,=).

change_pair_signature([[x-A,y-B],[x-C,y-D]|E],[F|G],=,=\=) :-
    !,
    A#=C#\B#\=D#<=>F,
    change_pair_signature([[x-C,y-D]|E],G,=,\=).

change_pair_signature([[x-A,y-B],[x-C,y-D]|E],[F|G],=,<) :-
    !,
    A#=C#\B#<D#<=>F,
    change_pair_signature([[x-C,y-D]|E],G,=,<).

change_pair_signature([[x-A,y-B],[x-C,y-D]|E],[F|G],=,>=) :-
    !,
    A#=C#\B#>=D#<=>F,
    change_pair_signature([[x-C,y-D]|E],G,=,>=).

change_pair_signature([[x-A,y-B],[x-C,y-D]|E],[F|G],=,>) :-
    !,
    A#=C#\B#>D#<=>F,
    change_pair_signature([[x-C,y-D]|E],G,=,>).

change_pair_signature([[x-A,y-B],[x-C,y-D]|E],[F|G],=,<=) :-

```



```

!,
A#<C#\B#<=>D#<=>F,
change_pair_signature ([ [x-C, y-D] | E], G, =, =< ) .

change_pair_signature ([ [x-A, y-B], [x-C, y-D] | E], [F|G], =\=, =) :-
!,
A#\<C#\B#<=>D#<=>F,
change_pair_signature ([ [x-C, y-D] | E], G, =\=, =) .

change_pair_signature ([ [x-A, y-B], [x-C, y-D] | E], [F|G], =\=, =\=) :-
!,
A#\<C#\B#\<=>D#\<=>F,
change_pair_signature ([ [x-C, y-D] | E], G, =\=, =\=) .

change_pair_signature ([ [x-A, y-B], [x-C, y-D] | E], [F|G], =\=, <) :-
!,
A#\<C#\B#<D#<=>F,
change_pair_signature ([ [x-C, y-D] | E], G, =\=, <) .

change_pair_signature ([ [x-A, y-B], [x-C, y-D] | E], [F|G], =\=, >=) :-
!,
A#\<C#\B#>=D#<=>F,
change_pair_signature ([ [x-C, y-D] | E], G, =\=, >=) .

change_pair_signature ([ [x-A, y-B], [x-C, y-D] | E], [F|G], =\=, >) :-
!,
A#\<C#\B#>D#<=>F,
change_pair_signature ([ [x-C, y-D] | E], G, =\=, >) .

change_pair_signature ([ [x-A, y-B], [x-C, y-D] | E], [F|G], =\=, =<) :-
!,
A#\<C#\B#<D#<=>F,
change_pair_signature ([ [x-C, y-D] | E], G, =\=, =<) .

change_pair_signature ([ [x-A, y-B], [x-C, y-D] | E], [F|G], <, =) :-
!,
A#<C#\B#<=>D#<=>F,
change_pair_signature ([ [x-C, y-D] | E], G, <, =) .

change_pair_signature ([ [x-A, y-B], [x-C, y-D] | E], [F|G], <, =\=) :-
!,
A#<C#\B#\<=>D#\<=>F,
change_pair_signature ([ [x-C, y-D] | E], G, <, =\=) .

change_pair_signature ([ [x-A, y-B], [x-C, y-D] | E], [F|G], <, <) :-
!,

```

```

A#<C#\B#<D#<=>F,
change_pair_signature ([ [x-C, y-D] | E], G, <, < ) .

change_pair_signature ([ [x-A, y-B], [x-C, y-D] | E], [F|G], <, >=) :-
!,
A#<C#\B#>=D#<=>F,
change_pair_signature ([ [x-C, y-D] | E], G, <, >=) .

change_pair_signature ([ [x-A, y-B], [x-C, y-D] | E], [F|G], <, >) :-
!,
A#<C#\B#>D#<=>F,
change_pair_signature ([ [x-C, y-D] | E], G, <, >) .

change_pair_signature ([ [x-A, y-B], [x-C, y-D] | E], [F|G], <, =<) :-
!,
A#<C#\B#=<D#<=>F,
change_pair_signature ([ [x-C, y-D] | E], G, <, =<) .

change_pair_signature ([ [x-A, y-B], [x-C, y-D] | E], [F|G], >=, =) :-
!,
A#>=C#\B#=D#<=>F,
change_pair_signature ([ [x-C, y-D] | E], G, >=, =) .

change_pair_signature ([ [x-A, y-B], [x-C, y-D] | E], [F|G], >=, =\=) :-
!,
A#>=C#\B#\=D#<=>F,
change_pair_signature ([ [x-C, y-D] | E], G, >=, =\=) .

change_pair_signature ([ [x-A, y-B], [x-C, y-D] | E], [F|G], >=, <) :-
!,
A#>=C#\B#<D#<=>F,
change_pair_signature ([ [x-C, y-D] | E], G, >=, <) .

change_pair_signature ([ [x-A, y-B], [x-C, y-D] | E], [F|G], >=, >=) :-
!,
A#>=C#\B#>=D#<=>F,
change_pair_signature ([ [x-C, y-D] | E], G, >=, >=) .

change_pair_signature ([ [x-A, y-B], [x-C, y-D] | E], [F|G], >=, >) :-
!,
A#>=C#\B#>D#<=>F,
change_pair_signature ([ [x-C, y-D] | E], G, >=, >) .

change_pair_signature ([ [x-A, y-B], [x-C, y-D] | E], [F|G], >=, =<) :-
!,
A#>=C#\B#=<D#<=>F,

```

```

change_pair_signature ([ [x-C, y-D] | E], G, >=, =< ) .

change_pair_signature ([ [x-A, y-B], [x-C, y-D] | E], [F|G], >, =) :-
!,
A#>C#\B#=D#<=>F,
change_pair_signature ([ [x-C, y-D] | E], G, >, =) .

change_pair_signature ([ [x-A, y-B], [x-C, y-D] | E], [F|G], >, =\=) :-
!,
A#>C#\B#\=D#<=>F,
change_pair_signature ([ [x-C, y-D] | E], G, >, =\=) .

change_pair_signature ([ [x-A, y-B], [x-C, y-D] | E], [F|G], >, <) :-
!,
A#>C#\B#<D#<=>F,
change_pair_signature ([ [x-C, y-D] | E], G, >, <) .

change_pair_signature ([ [x-A, y-B], [x-C, y-D] | E], [F|G], >, >=) :-
!,
A#>C#\B#>=D#<=>F,
change_pair_signature ([ [x-C, y-D] | E], G, >, >=) .

change_pair_signature ([ [x-A, y-B], [x-C, y-D] | E], [F|G], >, >) :-
!,
A#>C#\B#>D#<=>F,
change_pair_signature ([ [x-C, y-D] | E], G, >, >) .

change_pair_signature ([ [x-A, y-B], [x-C, y-D] | E], [F|G], >, =<) :-
!,
A#>C#\B#=<D#<=>F,
change_pair_signature ([ [x-C, y-D] | E], G, >, =<) .

change_pair_signature ([ [x-A, y-B], [x-C, y-D] | E], [F|G], =<, =) :-
!,
A#=<C#\B#=D#<=>F,
change_pair_signature ([ [x-C, y-D] | E], G, =<, =) .

change_pair_signature ([ [x-A, y-B], [x-C, y-D] | E], [F|G], =<, =\=) :-
!,
A#=<C#\B#\=D#<=>F,
change_pair_signature ([ [x-C, y-D] | E], G, =<, =\=) .

change_pair_signature ([ [x-A, y-B], [x-C, y-D] | E], [F|G], =<, <) :-
!,
A#=<C#\B#<D#<=>F,
change_pair_signature ([ [x-C, y-D] | E], G, =<, <) .

```

```

change_pair_signature([ [x-A,y-B], [x-C,y-D] | E], [F|G], =<, >=) :-
    !,
    A#=<C#\B#>=D#<=>F,
    change_pair_signature([ [x-C,y-D] | E], G, =<, >=) .

```

```

change_pair_signature([ [x-A,y-B], [x-C,y-D] | E], [F|G], =<, >) :-
    !,
    A#=<C#\B#>D#<=>F,
    change_pair_signature([ [x-C,y-D] | E], G, =<, >) .

```

```

change_pair_signature([ [x-A,y-B], [x-C,y-D] | E], [F|G], =<, =<) :-
    !,
    A#=<C#\B#=<D#<=>F,
    change_pair_signature([ [x-C,y-D] | E], G, =<, =<) .

```

B.37 change_partition

```
ctr_date(change_partition, ['20000128', '20030820', '20040530']).
```

```
ctr_origin(change_partition, 'Derived from %c.', [change]).
```

```
ctr_types(change_partition, ['VALUES'-collection(val-int)]).
```

```
ctr_arguments(
  change_partition,
  ['NCHANGE'-dvar,
   'VARIABLES'-collection(var-dvar),
   'PARTITIONS'-collection(p-'VALUES')]).
```

```
ctr_restrictions(
  change_partition,
  [required('VALUES', val),
   distinct('VALUES', val),
   'NCHANGE' >= 0,
   'NCHANGE' < size('VARIABLES'),
   required('VARIABLES', var),
   required('PARTITIONS', p),
   size('PARTITIONS') >= 2]).
```

```
ctr_graph(
  change_partition,
  ['VARIABLES'],
  2,
  ['PATH' >> collection(variables1, variables2)],
  [in_same_partition(
    variables1^var,
    variables2^var,
    'PARTITIONS')],
  ['NARC' = 'NCHANGE']).
```

```
ctr_example(
  change_partition,
  change_partition(
    2,
    [[var-6],
     [var-6],
     [var-2],
     [var-1],
     [var-3],
     [var-3],
     [var-1],
```

```
[var-6],  
[var-2],  
[var-2],  
[var-2]],  
[[p-[[val-1],[val-3]]],  
[p-[[val-4]]],  
[p-[[val-2],[val-6]]]])).
```

B.38 circuit

```

ctr_date(circuit, ['20030820', '20040530']).

ctr_origin(circuit, '\\cite{Lauriere78}', []).

ctr_synonyms(circuit, [atour, cycle]).

ctr_arguments(
    circuit,
    ['NODES'-collection(index-int, succ-dvar)]).

ctr_restrictions(
    circuit,
    [required('NODES', [index, succ]),
     'NODES'^index>=1,
     'NODES'^index<=size('NODES'),
     distinct('NODES', index),
     'NODES'^succ>=1,
     'NODES'^succ<=size('NODES')]).

ctr_graph(
    circuit,
    ['NODES'],
    2,
    ['CLIQUE'>>collection(nodes1, nodes2)],
    [nodes1^succ=nodes2^index],
    ['MIN_NSCC'=size('NODES'), 'MAX_ID'=1]).

ctr_example(
    circuit,
    circuit(
        [[index-1, succ-2],
         [index-2, succ-3],
         [index-3, succ-4],
         [index-4, succ-1]])).

```

B.39 circuit_cluster

```

ctr_date(circuit_cluster, ['20000128', '20030820']).

ctr_origin(
    circuit_cluster,
    'Inspired by \\cite{LaporteAsefVaziriSriskandarajah96}.',
    []).

ctr_arguments(
    circuit_cluster,
    ['NCIRCUIT'-dvar,
     'NODES'-collection(index-int, cluster-int, succ-dvar)]).

ctr_restrictions(
    circuit_cluster,
    ['NCIRCUIT'>=1,
     'NCIRCUIT'=<size('NODES'),
     required('NODES', [index, cluster, succ]),
     'NODES'^index>=1,
     'NODES'^index=<size('NODES'),
     distinct('NODES', index),
     'NODES'^succ>=1,
     'NODES'^succ=<size('NODES')]).

ctr_graph(
    circuit_cluster,
    ['NODES'],
    2,
    ['CLIQUE'>>collection(nodes1, nodes2)],
    [nodes1^succ=\=nodes1^index, nodes1^succ=nodes2^index],
    ['NTREE'=0, 'NSCC'='NCIRCUIT'],
    [>>('ALL_VERTICES',
        [-(variables,
            col('VARIABLES'-collection(var-dvar),
              [item(var-'NODES'^cluster)]))]),
    [alldifferent(variables),
     nvalues(variables, =, size('NODES', cluster))]).

ctr_example(
    circuit_cluster,
    [circuit_cluster(
        1,
        [[index-1, cluster-1, succ-1],
         [index-2, cluster-1, succ-4],
         [index-3, cluster-2, succ-3],

```



```
[index-4,cluster-2,succ-5],
[index-5,cluster-3,succ-8],
[index-6,cluster-3,succ-6],
[index-7,cluster-3,succ-7],
[index-8,cluster-4,succ-2],
[index-9,cluster-4,succ-9]]) ,
circuit_cluster(
  2,
  [[index-1,cluster-1,succ-1],
[index-2,cluster-1,succ-4],
[index-3,cluster-2,succ-3],
[index-4,cluster-2,succ-2],
[index-5,cluster-3,succ-5],
[index-6,cluster-3,succ-9],
[index-7,cluster-3,succ-7],
[index-8,cluster-4,succ-8],
[index-9,cluster-4,succ-6]])] .
```

B.40 circular_change

```

ctr_automaton(circular_change,circular_change).

ctr_date(circular_change,['20030820','20040530']).

ctr_origin(circular_change,'Derived from %c.',[change]).

ctr_arguments(
    circular_change,
    ['NCHANGE'-dvar,
     'VARIABLES'-collection(var-dvar),
     'CTR'-atom]).

ctr_restrictions(
    circular_change,
    ['NCHANGE'>=0,
     'NCHANGE'=<size('VARIABLES'),
     required('VARIABLES',var),
     in_list('CTR',[=,\=,<,>=,>,=<])]).

ctr_graph(
    circular_change,
    ['VARIABLES'],
    2,
    ['CIRCUIT'>>collection(variables1,variables2)],
    ['CTR'(variables1^var,variables2^var)],
    ['NARC'='NCHANGE']).

ctr_example(
    circular_change,
    circular_change(
        4,
        [[var-4],[var-4],[var-3],[var-4],[var-1]],
        =\=)).

circular_change(A,B,C) :-
    B=[D|E],
    append(B,[D],F),
    circular_change_signature(F,G,C),
    automaton(
        G,
        H,
        G,
        0..1,
        [source(s),sink(t)],

```

```

    [arc(s,0,s),arc(s,1,s,[I+1]),arc(s,$,t)],
    [I],
    [0],
    [A]).

```

```
circular_change_signature([],[],A).
```

```
circular_change_signature([A],[],B) :-
    !.
```

```
circular_change_signature([[var-A],[var-B]|C],[D|E],=) :-
    !,
    A#=B#<=>D,
    circular_change_signature([[var-B]|C],E,=).
```

```
circular_change_signature([[var-A],[var-B]|C],[D|E],=\=) :-
    !,
    A#\=B#<=>D,
    circular_change_signature([[var-B]|C],E,=\=).
```

```
circular_change_signature([[var-A],[var-B]|C],[D|E],<) :-
    !,
    A#<B#<=>D,
    circular_change_signature([[var-B]|C],E,<).
```

```
circular_change_signature([[var-A],[var-B]|C],[D|E],>=) :-
    !,
    A#>=B#<=>D,
    circular_change_signature([[var-B]|C],E,>=).
```

```
circular_change_signature([[var-A],[var-B]|C],[D|E],>) :-
    !,
    A#>B#<=>D,
    circular_change_signature([[var-B]|C],E,>).
```

```
circular_change_signature([[var-A],[var-B]|C],[D|E],=<) :-
    !,
    A#=<B#<=>D,
    circular_change_signature([[var-B]|C],E,=<).
```

B.41 clique

```

ctr_date(clique, ['20030820', '20040530']).

ctr_origin(clique, '\\cite{Fahle02}', []).

ctr_arguments(
    clique,
    ['SIZE_CLIQUE'-dvar,
     'NODES'-collection(index-int, succ-svar)]).

ctr_restrictions(
    clique,
    ['SIZE_CLIQUE'>=0,
     'SIZE_CLIQUE'<=size('NODES'),
     required('NODES', [index, succ]),
     'NODES'^index>=1,
     'NODES'^index<=size('NODES'),
     distinct('NODES', index)]).

ctr_graph(
    clique,
    ['NODES'],
    2,
    ['CLIQUE' (=\\=)>>collection(nodes1, nodes2)],
    [in_set(nodes2^index, nodes1^succ)],
    ['NARC'=' SIZE_CLIQUE' *' SIZE_CLIQUE' -' SIZE_CLIQUE',
     'NVERTEX'=' SIZE_CLIQUE']).

ctr_example(
    clique,
    clique(
        3,
        [[index-1, succ-{}],
         [index-2, succ-{3, 5}],
         [index-3, succ-{2, 5}],
         [index-4, succ-{}],
         [index-5, succ-{2, 3}]])).

```

B.42 colored_matrix

```

ctr_predefined(colored_matrix).

ctr_date(colored_matrix, ['20031017', '20040530']).

ctr_origin(colored_matrix, 'KOALOG', []).

ctr_synonyms(colored_matrix, [cardinality_matrix, card_matrix]).

ctr_arguments(
    colored_matrix,
    ['C'-int,
     'L'-int,
     'K'-int,
     'MATRIX'-collection(column-int, line-int, var-dvar),
     'CPROJ'-collection(column-int, val-int, noccurrence-dvar),
     'LPROJ'-collection(line-int, val-int, noccurrence-dvar)].

ctr_restrictions(
    colored_matrix,
    ['C'>=0,
     'L'>=0,
     'K'>=0,
     required('MATRIX', [column, line, var]),
     increasing_seq('MATRIX', [column, line]),
     size('MATRIX')='C'*'L'+ 'C'+ 'L'+1,
     'MATRIX'^column>=0,
     'MATRIX'^column=<'C',
     'MATRIX'^line>=0,
     'MATRIX'^line=<'L',
     'MATRIX'^var>=0,
     'MATRIX'^var=<'K',
     required('CPROJ', [column, val, noccurrence]),
     increasing_seq('CPROJ', [column, val]),
     size('CPROJ')='C'*'K'+ 'C'+ 'K'+1,
     'CPROJ'^column>=0,
     'CPROJ'^column=<'C',
     'CPROJ'^val>=0,
     'CPROJ'^val=<'K',
     required('LPROJ', [line, val, noccurrence]),
     increasing_seq('LPROJ', [line, val]),
     size('LPROJ')='L'*'K'+ 'L'+ 'K'+1,
     'LPROJ'^line>=0,
     'LPROJ'^line=<'L',
     'LPROJ'^val>=0,

```

```

'LPROJ'^val=<'K'] ).

ctr_example(
  colored_matrix,
  colored_matrix(
    1,
    2,
    4,
    [[column-0,line-0,var-3],
     [column-0,line-1,var-1],
     [column-0,line-2,var-3],
     [column-1,line-0,var-4],
     [column-1,line-1,var-4],
     [column-1,line-2,var-3]],
    [[column-0,val-0,nocc-0],
     [column-0,val-1,nocc-1],
     [column-0,val-2,nocc-0],
     [column-0,val-3,nocc-2],
     [column-0,val-4,nocc-0],
     [column-1,val-0,nocc-0],
     [column-1,val-1,nocc-0],
     [column-1,val-2,nocc-0],
     [column-1,val-3,nocc-1],
     [column-1,val-4,nocc-2]],
    [[line-0,val-0,nocc-0],
     [line-0,val-1,nocc-0],
     [line-0,val-2,nocc-0],
     [line-0,val-3,nocc-1],
     [line-0,val-4,nocc-1],
     [line-1,val-0,nocc-0],
     [line-1,val-1,nocc-1],
     [line-1,val-2,nocc-0],
     [line-1,val-3,nocc-0],
     [line-1,val-4,nocc-1],
     [line-2,val-0,nocc-0],
     [line-2,val-1,nocc-0],
     [line-2,val-2,nocc-0],
     [line-2,val-3,nocc-2],
     [line-2,val-4,nocc-0]])).

```

B.43 coloured_cumulative

```
ctr_date(coloured_cumulative, ['20000128', '20030820']).
```

```
ctr_origin(
  coloured_cumulative,
  'Derived from %c and %c.',
  [cumulative, nvalues]).
```

```
ctr_arguments(
  coloured_cumulative,
  [-( 'TASKS',
      collection(
        origin-dvar,
        duration-dvar,
        end-dvar,
        colour-dvar)),
    'LIMIT'-int]).
```

```
ctr_restrictions(
  coloured_cumulative,
  [require_at_least(2, 'TASKS', [origin, duration, end]),
   required('TASKS', colour),
   'TASKS'^duration>=0,
   'LIMIT'>=0]).
```

```
ctr_graph(
  coloured_cumulative,
  ['TASKS'],
  1,
  ['SELF'>>collection(tasks)],
  [tasks^origin+tasks^duration=tasks^end],
  ['NARC'=size('TASKS')]).
```

```
ctr_graph(
  coloured_cumulative,
  ['TASKS', 'TASKS'],
  2,
  ['PRODUCT'>>collection(tasks1, tasks2)],
  [tasks1^duration>0,
   tasks2^origin=<tasks1^origin,
   tasks1^origin<tasks2^end],
  [],
  [>>('SUCC',
      [source,
       -(variables,
```

```
col('VARIABLES'-collection(var-dvar),
    [item(var-'TASKS'^colour)])]),
[nvalues(variables,=<,'LIMIT')]).

ctr_example(
  coloured_cumulative,
  coloured_cumulative(
    [[origin-1,duration-2,end-3,colour-1],
     [origin-2,duration-9,end-11,colour-2],
     [origin-3,duration-10,end-13,colour-3],
     [origin-6,duration-6,end-12,colour-2],
     [origin-7,duration-2,end-9,colour-3]],
    2)).
```


B.44 coloured_cumulatives

```

ctr_date(coloured_cumulatives, ['20000128', '20030820']).

ctr_origin(
    coloured_cumulatives,
    'Derived from %c and %c.',
    [cumulatives, nvalues]).

ctr_arguments(
    coloured_cumulatives,
    [-( 'TASKS',
        collection(
            machine-dvar,
            origin-dvar,
            duration-dvar,
            end-dvar,
            colour-dvar)),
        'MACHINES'-collection(id-int, capacity-int)]).

ctr_restrictions(
    coloured_cumulatives,
    [required('TASKS', [machine, colour]),
     require_at_least(2, 'TASKS', [origin, duration, end]),
     'TASKS'^duration>=0,
     required('MACHINES', [id, capacity]),
     distinct('MACHINES', id),
     'MACHINES'^capacity>=0]).

ctr_graph(
    coloured_cumulatives,
    ['TASKS'],
    1,
    ['SELF'>>collection(tasks)],
    [tasks^origin+tasks^duration=tasks^end],
    ['NARC'=size('TASKS')]).

ctr_graph(
    coloured_cumulatives,
    ['TASKS', 'TASKS'],
    2,
    foreach('MACHINES', ['PRODUCT'>>collection(tasks1, tasks2)]),
    [tasks1^machine='MACHINES'^id,
     tasks1^machine=tasks2^machine,
     tasks1^duration>0,
     tasks2^origin=<tasks1^origin,

```

```

    tasks1^origin<tasks2^end],
  [],
  [>>('SUCC',
    [source,
      -(variables,
        col('VARIABLES'-collection(var-dvar),
          [item(var-'TASKS'^colour)]))]],
    [nvalues(variables,=<,'MACHINES'^capacity)]).

ctr_example(
  coloured_cumulatives,
  coloured_cumulatives(
    [[machine-1,origin-6,duration-6,end-12,colour-1],
     [machine-1,origin-2,duration-9,end-11,colour-2],
     [machine-2,origin-7,duration-3,end-10,colour-2],
     [machine-1,origin-1,duration-2,end-3,colour-1],
     [machine-2,origin-4,duration-5,end-9,colour-2],
     [machine-1,origin-3,duration-10,end-13,colour-1]],
    [[id-1,capacity-2],[id-2,capacity-1]]).

```

B.45 common

```

ctr_date(common, ['20000128', '20030820']).

ctr_origin(common, 'N.~Beldiceanu', []).

ctr_arguments(
    common,
    ['NCOMMON1'-dvar,
     'NCOMMON2'-dvar,
     'VARIABLES1'-collection(var-dvar),
     'VARIABLES2'-collection(var-dvar)]).

ctr_restrictions(
    common,
    ['NCOMMON1'>=0,
     'NCOMMON1'=<size('VARIABLES1'),
     'NCOMMON2'>=0,
     'NCOMMON2'=<size('VARIABLES2'),
     required('VARIABLES1', var),
     required('VARIABLES2', var)]).

ctr_graph(
    common,
    ['VARIABLES1', 'VARIABLES2'],
    2,
    ['PRODUCT'>>collection(variables1, variables2)],
    [variables1^var=variables2^var],
    ['NSOURCE'='NCOMMON1', 'NSINK'='NCOMMON2']).

ctr_example(
    common,
    common(
        3,
        4,
        [[var-1], [var-9], [var-1], [var-5]],
        [[var-2], [var-1], [var-9], [var-9], [var-6], [var-9]])).

```

B.46 common_interval

```

ctr_date(common_interval, ['20030820']).

ctr_origin(common_interval, 'Derived from %c.', [common]).

ctr_arguments(
    common_interval,
    ['NCOMMON1'-dvar,
     'NCOMMON2'-dvar,
     'VARIABLES1'-collection(var-dvar),
     'VARIABLES2'-collection(var-dvar),
     'SIZE_INTERVAL'-int]).

ctr_restrictions(
    common_interval,
    ['NCOMMON1'>=0,
     'NCOMMON1'=<size('VARIABLES1'),
     'NCOMMON2'>=0,
     'NCOMMON2'=<size('VARIABLES2'),
     required('VARIABLES1', var),
     required('VARIABLES2', var),
     'SIZE_INTERVAL'>0]).

ctr_graph(
    common_interval,
    ['VARIABLES1', 'VARIABLES2'],
    2,
    ['PRODUCT'>>collection(variables1, variables2)],
    [(variables1^var/'SIZE_INTERVAL',
      variables2^var/'SIZE_INTERVAL')],
    ['NSOURCE'='NCOMMON1', 'NSINK'='NCOMMON2']).

ctr_example(
    common_interval,
    common_interval(
        3,
        2,
        [[var-8], [var-6], [var-6], [var-0]],
        [[var-7], [var-3], [var-3], [var-3], [var-3], [var-7]],
        3)).

```

B.47 common_modulo

```

ctr_date(common_modulo, ['20030820']).

ctr_origin(common_modulo, 'Derived from %c.', [common]).

ctr_arguments(
    common_modulo,
    ['NCOMMON1'-dvar,
     'NCOMMON2'-dvar,
     'VARIABLES1'-collection(var-dvar),
     'VARIABLES2'-collection(var-dvar),
     'M'-int]).

ctr_restrictions(
    common_modulo,
    ['NCOMMON1'>=0,
     'NCOMMON1'=<size('VARIABLES1'),
     'NCOMMON2'>=0,
     'NCOMMON2'=<size('VARIABLES2'),
     required('VARIABLES1', var),
     required('VARIABLES2', var),
     'M'>0]).

ctr_graph(
    common_modulo,
    ['VARIABLES1', 'VARIABLES2'],
    2,
    ['PRODUCT'>>collection(variables1, variables2)],
    [variables1^var mod 'M'=variables2^var mod 'M'],
    ['NSOURCE'='NCOMMON1', 'NSINK'='NCOMMON2']).

ctr_example(
    common_modulo,
    common_modulo(
        3,
        4,
        [[var-0], [var-4], [var-0], [var-8]],
        [[var-7], [var-5], [var-4], [var-9], [var-2], [var-4]],
        5)).

```

B.48 common_partition

```

ctr_date(common_partition, ['20030820']).

ctr_origin(common_partition, 'Derived from %c.', [common]).

ctr_types(common_partition, ['VALUES'-collection(val-int)]).

ctr_arguments(
    common_partition,
    ['NCOMMON1'-dvar,
     'NCOMMON2'-dvar,
     'VARIABLES1'-collection(var-dvar),
     'VARIABLES2'-collection(var-dvar),
     'PARTITIONS'-collection(p-'VALUES')]).

ctr_restrictions(
    common_partition,
    [required('VALUES', val),
     distinct('VALUES', val),
     'NCOMMON1'>=0,
     'NCOMMON1'=<size('VARIABLES1'),
     'NCOMMON2'>=0,
     'NCOMMON2'=<size('VARIABLES2'),
     required('VARIABLES1', var),
     required('VARIABLES2', var),
     required('PARTITIONS', p),
     size('PARTITIONS')>=2]).

ctr_graph(
    common_partition,
    ['VARIABLES1', 'VARIABLES2'],
    2,
    ['PRODUCT'>>collection(variables1, variables2)],
    [in_same_partition(
        variables1^var,
        variables2^var,
        'PARTITIONS')],
    ['NSOURCE'='NCOMMON1', 'NSINK'='NCOMMON2']).

ctr_example(
    common_partition,
    common_partition(
        3,
        4,
        [[var-2], [var-3], [var-6], [var-0]],

```

```
[[var-0],[var-6],[var-3],[var-3],[var-7],[var-1]],  
[[p-[[val-1],[val-3]]],  
[p-[[val-4]]],  
[p-[[val-2],[val-6]]]])).
```

B.49 connect_points

```
ctr_date(connect_points, ['20000128', '20030820', '20040530']).
```

```
ctr_origin(connect_points, 'N.~Beldiceanu', []).
```

```
ctr_arguments(
    connect_points,
    ['SIZE1'-int,
     'SIZE2'-int,
     'SIZE3'-int,
     'NGROUP'-dvar,
     'POINTS'-collection(p-dvar)]).
```

```
ctr_restrictions(
    connect_points,
    ['SIZE1'>0,
     'SIZE2'>0,
     'SIZE3'>0,
     'NGROUP'>=0,
     'NGROUP'=<size('POINTS'),
     'SIZE1'*'SIZE2'*'SIZE3'=size('POINTS'),
     required('POINTS',p)]).
```

```
ctr_graph(
    connect_points,
    ['POINTS'],
    2,
    [>>('GRID'(['SIZE1','SIZE2','SIZE3']),
         collection(points1,points2))],
    [points1^p=\=0,points1^p=points2^p],
    ['NSCC'='NGROUP']).
```

```
ctr_example(
    connect_points,
    connect_points(
        8,
        4,
        2,
        2,
        [[p-0],
         [p-0],
         [p-1],
         [p-1],
         [p-0],
         [p-2],
```


[p-2],
[p-2],
[p-0],
[p-0],
[p-0],
[p-2],
[p-0],
[p-0],
[p-0],
[p-2],
[p-0],
[p-0]])).

B.50 correspondence

```

ctr_date(correspondence, ['20030820']).

ctr_origin(
  correspondence,
  'Derived from %c by removing the sorting condition.',
  [sort_permutation]).

ctr_arguments(
  correspondence,
  ['FROM'-collection(fvar-dvar),
   'PERMUTATION'-collection(var-dvar),
   'TO'-collection(tvar-dvar)]).

ctr_restrictions(
  correspondence,
  [size('PERMUTATION')==size('FROM'),
   size('PERMUTATION')==size('TO'),
   'PERMUTATION'^var>=1,
   'PERMUTATION'^var<=size('PERMUTATION'),
   alldifferent('PERMUTATION'),
   required('FROM', fvar),
   required('PERMUTATION', var),
   required('TO', tvar)]).

ctr_derived_collections(
  correspondence,
  [col('FROM_PERMUTATION'-collection(fvar-dvar, var-dvar),
      [item(fvar-'FROM'^fvar, var-'PERMUTATION'^var)])]).

ctr_graph(
  correspondence,
  ['FROM_PERMUTATION', 'TO'],
  2,
  ['PRODUCT'>>collection(from_permutation, to)],
  [from_permutation^fvar=to^tvar,
   from_permutation^var=to^key],
  ['NARC'=size('PERMUTATION')]).

ctr_example(
  correspondence,
  correspondence(
    [[fvar-1],
     [fvar-9],
     [fvar-1],

```

```
[fvar-5],  
[fvar-2],  
[fvar-1]],  
[[var-6],[var-1],[var-3],[var-5],[var-4],[var-2]],  
[[tvar-9],  
[tvar-1],  
[tvar-1],  
[tvar-2],  
[tvar-5],  
[tvar-1]])).
```

B.51 count

```

ctr_automaton(count, count_).

ctr_date(count, ['20000128', '20030820', '20040530']).

ctr_origin(count, '\\cite{Sicstus95}', []).

ctr_arguments(
  count,
  ['VALUE'-int,
   'VARIABLES'-collection(var-dvar),
   'RELOP'-atom,
   'NVAR'-dvar]).

ctr_restrictions(
  count,
  [required('VARIABLES', var),
   in_list('RELOP', [=, =\=, <, >=, >, =<])]).

ctr_graph(
  count,
  ['VARIABLES'],
  1,
  ['SELF'>>collection(variables)],
  [variables^var='VALUE'],
  ['RELOP'('NARC', 'NVAR')]).

ctr_example(
  count,
  count(5, [[var-4], [var-5], [var-5], [var-4], [var-5]], >=, 2)).

count_(A,B,C,D) :-
  length(B,E),
  in(F,0..E),
  count_signature(B,G,A),
  automaton(
    G,
    H,
    G,
    0..1,
    [source(s), sink(t)],
    [arc(s,0,s), arc(s,1,s,[I+1]), arc(s,$,t)],
    [I],
    [0],
    [F]),

```

```
count_relop(C,F,D) .

count_signature([],[],A) .

count_signature([[var-A]|B],[C|D],E) :-
    A#=E#<=>C,
    count_signature(B,D,E) .

count_relop(=,A,B) :-
    A#=B.

count_relop(=\=,A,B) :-
    A#\=B.

count_relop(<,A,B) :-
    A#<B.

count_relop(>=,A,B) :-
    A#>=B.

count_relop(>,A,B) :-
    A#>B.

count_relop(=<,A,B) :-
    A#=<B.
```

B.52 counts

```

ctr_automaton(counts, counts) .

ctr_date(counts, ['20030820', '20040530']) .

ctr_origin(counts, 'Derived from %c.', [count]) .

ctr_arguments(
  counts,
  ['VALUES'-collection(val-int),
   'VARIABLES'-collection(var-dvar),
   'RELOP'-atom,
   'LIMIT'-dvar]) .

ctr_restrictions(
  counts,
  [required('VALUES', val),
   distinct('VALUES', val),
   required('VARIABLES', var),
   in_list('RELOP', [=,=\=, <, >=, >, =<])]) .

ctr_graph(
  counts,
  ['VARIABLES', 'VALUES'],
  2,
  ['PRODUCT'>>collection(variables, values)],
  [variables^var=values^val],
  ['RELOP'('NARC', 'LIMIT')]) .

ctr_example(
  counts,
  counts(
    [[val-1], [val-3], [val-4], [val-9]],
    [[var-4], [var-5], [var-5], [var-4], [var-1], [var-5]],
    =,
    3)) .

counts(A, B, C, D) :-
  length(B, E),
  in(F, 0..E),
  col_to_list(A, G),
  list_to_fdset(G, H),
  counts_signature(B, I, H),
  automaton(
    I,

```

```
J,  
I,  
0..1,  
[source(s),sink(t)],  
[arc(s,0,s),arc(s,1,s,[K+1]),arc(s,$,t)],  
[K],  
[0],  
[F]),  
count_relop(C,F,D).  
  
counts_signature([],[],A).  
  
counts_signature([[var-A]|B],[C|D],E) :-  
in_set(A,E)#<=>C,  
counts_signature(B,D,E).
```


B.53 crossing

```

ctr_date(crossing, ['20000128', '20030820']).

ctr_origin(
  crossing,
  'Inspired by \\cite{CormenLeisersonRivest90}.'.
  []).

ctr_arguments(
  crossing,
  ['NCROSS'-dvar,
   'SEGMENTS'-collection(ox-dvar,oy-dvar,ex-dvar,ey-dvar)]).

ctr_restrictions(
  crossing,
  ['NCROSS'>=0,
   =<('NCROSS',
      /(-(size('SEGMENTS')*size('SEGMENTS'),
          size('SEGMENTS'))),
      2)),
   required('SEGMENTS', [ox,oy,ex,ey])].

ctr_graph(
  crossing,
  ['SEGMENTS'],
  2,
  ['CLIQUE' (<)>>collection(s1,s2)],
  [max(s1^ox,s1^ex)>=min(s2^ox,s2^ex),
   max(s2^ox,s2^ex)>=min(s1^ox,s1^ex),
   max(s1^oy,s1^ey)>=min(s2^oy,s2^ey),
   max(s2^oy,s2^ey)>=min(s1^oy,s1^ey),
   #\/#\/#\/(=-(s2^ox-s1^ex)*(s1^ey-s1^oy),
              (s1^ex-s1^ox)*(s2^oy-s1^ey)),
              0),
              =-(s2^ex-s1^ex)*(s2^oy-s1^oy),
              (s2^ox-s1^ox)*(s2^ey-s1^ey)),
              0)),
   =\=(sign(
              -(s2^ox-s1^ex)*(s1^ey-s1^oy),
              (s1^ex-s1^ox)*(s2^oy-s1^ey))),
         sign(
              -(s2^ex-s1^ex)*(s2^oy-s1^oy),
              (s2^ox-s1^ox)*(s2^ey-s1^ey))))],
  ['NARC'='NCROSS']).

```

```
ctr_example(  
  crossing,  
  crossing(  
    3,  
    [[ox-1,oy-4,ex-9,ey-2],  
     [ox-1,oy-1,ex-3,ey-5],  
     [ox-3,oy-2,ex-7,ey-4],  
     [ox-9,oy-1,ex-9,ey-4]])).
```

B.54 cumulative

```
ctr_date(cumulative, ['20000128', '20030820', '20040530']).
```

```
ctr_origin(cumulative, '\\cite{AggounBeldiceanu93}', []).
```

```
ctr_arguments(
  cumulative,
  [-( 'TASKS',
      collection(
        origin-dvar,
        duration-dvar,
        end-dvar,
        height-dvar)),
    'LIMIT'-int]).
```

```
ctr_restrictions(
  cumulative,
  [require_at_least(2, 'TASKS', [origin, duration, end]),
   required('TASKS', height),
   'TASKS'^duration>=0,
   'TASKS'^height>=0,
   'LIMIT'>=0]).
```

```
ctr_graph(
  cumulative,
  ['TASKS'],
  1,
  ['SELF'>>collection(tasks)],
  [tasks^origin+tasks^duration=tasks^end],
  ['NARC'=size('TASKS')]).
```

```
ctr_graph(
  cumulative,
  ['TASKS', 'TASKS'],
  2,
  ['PRODUCT'>>collection(tasks1, tasks2)],
  [tasks1^duration>0,
   tasks2^origin<tasks1^origin,
   tasks1^origin<tasks2^end],
  [],
  [>>('SUCC',
      [source,
       -(variables,
         col('VARIABLES'-collection(var-dvar),
            [item(var-'TASKS'^height)]))]]],
```

```
[sum_ctr(variables,=<,'LIMIT')]).  
  
ctr_example(  
  cumulative,  
  cumulative(  
    [[origin-1,duration-3,end-4,height-1],  
     [origin-2,duration-9,end-11,height-2],  
     [origin-3,duration-10,end-13,height-1],  
     [origin-6,duration-6,end-12,height-1],  
     [origin-7,duration-2,end-9,height-3]],  
    8)).
```

B.55 cumulative_product

```

ctr_date(cumulative_product, ['20030820']).

ctr_origin(cumulative_product, 'Derived from %c.', [cumulative]).

ctr_arguments(
    cumulative_product,
    [-( 'TASKS',
        collection(
            origin-dvar,
            duration-dvar,
            end-dvar,
            height-dvar)),
    'LIMIT'-int]).

ctr_restrictions(
    cumulative_product,
    [require_at_least(2, 'TASKS', [origin, duration, end]),
    required('TASKS', height),
    'TASKS'^duration>=0,
    'TASKS'^height>=1,
    'LIMIT'>=0]).

ctr_graph(
    cumulative_product,
    ['TASKS'],
    1,
    ['SELF'>>collection(tasks)],
    [tasks^origin+tasks^duration=tasks^end],
    ['NARC'=size('TASKS')]).

ctr_graph(
    cumulative_product,
    ['TASKS', 'TASKS'],
    2,
    ['PRODUCT'>>collection(tasks1, tasks2)],
    [tasks1^duration>0,
    tasks2^origin=<tasks1^origin,
    tasks1^origin<tasks2^end],
    [],
    [>>('SUCC',
        [source,
        -(variables,
            col('VARIABLES'-collection(var-dvar),
                [item(var-'ITEMS'^height)]))]]],

```

```
[product_ctr(variables,=<,'LIMIT')]).  
  
ctr_example(  
  cumulative_product,  
  cumulative_product(  
    [[origin-1,duration-3,end-4,height-1],  
     [origin-2,duration-9,end-11,height-2],  
     [origin-3,duration-10,end-13,height-1],  
     [origin-6,duration-6,end-12,height-1],  
     [origin-7,duration-2,end-9,height-3]],  
    6)).
```

B.56 cumulative_two_d

```

ctr_date(cumulative_two_d, ['20000128', '20030820']).

ctr_origin(
    cumulative_two_d,
    'Inspired by %c and %c.',
    [cumulative, diffn]).

ctr_arguments(
    cumulative_two_d,
    [-( 'RECTANGLES',
        collection(
            start1-dvar,
            size1-dvar,
            last1-dvar,
            start2-dvar,
            size2-dvar,
            last2-dvar,
            height-dvar)),
    'LIMIT'-int]).

ctr_restrictions(
    cumulative_two_d,
    [require_at_least(2, 'RECTANGLES', [start1, size1, last1]),
    require_at_least(2, 'RECTANGLES', [start2, size2, last2]),
    required('RECTANGLES', height),
    'RECTANGLES' ^ size1 >= 0,
    'RECTANGLES' ^ size2 >= 0,
    'RECTANGLES' ^ height >= 0,
    'LIMIT' >= 0]).

ctr_derived_collections(
    cumulative_two_d,
    [col( -( 'CORNERS',
            collection(size1-dvar, size2-dvar, x-dvar, y-dvar)),
    [item(
        size1-'RECTANGLES' ^ size1,
        size2-'RECTANGLES' ^ size2,
        x-'RECTANGLES' ^ start1,
        y-'RECTANGLES' ^ start2),
    item(
        size1-'RECTANGLES' ^ size1,
        size2-'RECTANGLES' ^ size2,
        x-'RECTANGLES' ^ start1,
        y-'RECTANGLES' ^ last2),

```

```

        item(
            size1-'RECTANGLES'^size1,
            size2-'RECTANGLES'^size2,
            x-'RECTANGLES'^last1,
            y-'RECTANGLES'^start2),
        item(
            size1-'RECTANGLES'^size1,
            size2-'RECTANGLES'^size2,
            x-'RECTANGLES'^last1,
            y-'RECTANGLES'^last2)]])).

ctr_graph(
    cumulative_two_d,
    ['RECTANGLES'],
    1,
    ['SELF'>>collection(rectangles)],
    [rectangles^start1+rectangles^size1-1=rectangles^last1,
     rectangles^start2+rectangles^size2-1=rectangles^last2],
    ['NARC'=size('RECTANGLES')]).

ctr_graph(
    cumulative_two_d,
    ['CORNERS','RECTANGLES'],
    2,
    ['PRODUCT'>>collection(corners,rectangles)],
    [corners^size1>0,
     corners^size2>0,
     rectangles^start1=<corners^x,
     corners^x=<rectangles^last1,
     rectangles^start2=<corners^y,
     corners^y=<rectangles^last2],
    [],
    [>>('SUCC',
        [source,
         -(variables,
            col('VARIABLES'-collection(var-dvar),
                [item(var-'RECTANGLES'^height)]))]),
     [sum_ctr(variables,=<,'LIMIT')]).

ctr_example(
    cumulative_two_d,
    cumulative_two_d(
        [start1-1,
         size1-4,
         last1-4,
         start2-3,
```



```
size2-3,  
last2-5,  
height-4],  
[start1-3,  
size1-2,  
last1-4,  
start2-1,  
size2-2,  
last2-2,  
height-2],  
[start1-1,  
size1-2,  
last1-2,  
start2-1,  
size2-2,  
last2-2,  
height-3],  
[start1-4,  
size1-1,  
last1-4,  
start2-1,  
size2-1,  
last2-1,  
height-1]],  
4) ).
```

B.57 cumulative_with_level_of_priority

```

ctr_date(cumulative_with_level_of_priority, ['20040530']).

ctr_origin(cumulative_with_level_of_priority, 'H.~Simonis', []).

ctr_arguments(
  cumulative_with_level_of_priority,
  [-( 'TASKS',
      collection(
        priority-int,
        origin-dvar,
        duration-dvar,
        end-dvar,
        height-dvar),
      'PRIORITIES'-collection(id-int, capacity-int)]).

ctr_restrictions(
  cumulative_with_level_of_priority,
  [required('TASKS', [priority, height]),
   require_at_least(2, 'TASKS', [origin, duration, end]),
   'TASKS' ^priority >= 1,
   'TASKS' ^priority <= size('PRIORITIES'),
   'TASKS' ^duration >= 0,
   'TASKS' ^height >= 0,
   required('PRIORITIES', [id, capacity]),
   'PRIORITIES' ^id >= 1,
   'PRIORITIES' ^id <= size('PRIORITIES'),
   increasing_seq('PRIORITIES', id),
   increasing_seq('PRIORITIES', capacity)]).

ctr_derived_collections(
  cumulative_with_level_of_priority,
  [col(-( 'TIME_POINTS',
          collection(idp-int, duration-dvar, point-dvar)),
       [item(
          idp-'TASKS' ^priority,
          duration-'TASKS' ^duration,
          point-'TASKS' ^origin),
        item(
          idp-'TASKS' ^priority,
          duration-'TASKS' ^duration,
          point-'TASKS' ^end)])]).

ctr_graph(
  cumulative_with_level_of_priority,

```

```

['TASKS'],
1,
['SELF'>>collection(tasks)],
[tasks^origin+tasks^duration=tasks^end],
['NARC'=size('TASKS')]).

ctr_graph(
cumulative_with_level_of_priority,
['TIME_POINTS','TASKS'],
2,
foreach(
'PRIORITIES',
['PRODUCT'>>collection(time_points,tasks)]),
[time_points^idp='PRIORITIES'^id,
time_points^idp>=tasks^priority,
time_points^duration>0,
tasks^origin<time_points^point,
time_points^point<tasks^end],
[],
[>>('SUCC',
[source,
-(variables,
col('VARIABLES'-collection(var-dvar),
[item(var-'TASKS'^height)])])],
[sum_ctr(variables,=<,'PRIORITIES'^capacity)])].

ctr_example(
cumulative_with_level_of_priority,
cumulative_with_level_of_priority(
[[priority-1,origin-1,duration-2,end-3,height-1],
[priority-1,origin-2,duration-3,end-5,height-1],
[priority-1,origin-5,duration-2,end-7,height-2],
[priority-2,origin-3,duration-2,end-5,height-2],
[priority-2,origin-6,duration-3,end-9,height-1]],
[[id-1,capacity-2],[id-2,capacity-3]])).

```

B.58 cumulatives

```

ctr_date(cumulatives, ['20000128', '20030820', '20040530']).

ctr_origin(cumulatives, '\\cite{BeldiceanuCarlsson02a}', []).

ctr_arguments(
  cumulatives,
  [-( 'TASKS',
      collection(
        machine-dvar,
        origin-dvar,
        duration-dvar,
        end-dvar,
        height-dvar)),
    'MACHINES'-collection(id-int, capacity-int),
    'CTR'-atom]).

ctr_restrictions(
  cumulatives,
  [required('TASKS', [machine, height]),
   require_at_least(2, 'TASKS', [origin, duration, end]),
   in_attr('TASKS', machine, 'MACHINES', id),
   'TASKS'^duration>=0,
   required('MACHINES', [id, capacity]),
   distinct('MACHINES', id),
   in_list('CTR', [=,<, >=])]).

ctr_derived_collections(
  cumulatives,
  [col(-( 'TIME_POINTS',
          collection(idm-int, duration-dvar, point-dvar)),
        [item(
            idm-'TASKS'^machine,
            duration-'TASKS'^duration,
            point-'TASKS'^origin),
         item(
            idm-'TASKS'^machine,
            duration-'TASKS'^duration,
            point-'TASKS'^end)])]).

ctr_graph(
  cumulatives,
  ['TASKS'],
  1,
  ['SELF'>>collection(tasks)],

```

```

[tasks^origin+tasks^duration=tasks^end],
['NARC'=size('TASKS')]).

ctr_graph(
  cumulatives,
  ['TIME_POINTS','TASKS'],
  2,
  foreach(
    'MACHINES',
    ['PRODUCT'>>collection(time_points,tasks)]),
  [time_points^idm='MACHINES'^id,
  time_points^idm=tasks^machine,
  time_points^duration>0,
  tasks^origin=<time_points^point,
  time_points^point<tasks^end],
  [],
  [>>('SUCC',
    [source,
    -(variables,
    col('VARIABLES'-collection(var-dvar),
    [item(var-'TASKS'^height)])])],
  [sum_ctr(variables,'CTR','MACHINES'^capacity)]).

ctr_example(
  cumulatives,
  cumulatives(
    [[machine-1,origin-2,duration-2,end-4,height- -2],
    [machine-1,origin-1,duration-4,end-5,height-1],
    [machine-1,origin-4,duration-2,end-6,height- -1],
    [machine-1,origin-2,duration-3,end-5,height-2],
    [machine-1,origin-5,duration-2,end-7,height-2],
    [machine-2,origin-3,duration-2,end-5,height- -1],
    [machine-2,origin-1,duration-4,end-5,height-1]],
    [[id-1,capacity-0],[id-2,capacity-0]],
    >=)).

```

B.59 cutset

```

ctr_date(cutset, ['20030820', '20040530']).

ctr_origin(cutset, '\\cite{FagesLal03}', []).

ctr_arguments(
  cutset,
  ['SIZE_CUTSET'-dvar,
   'NODES'-collection(index-int, succ-sint, bool-dvar)]).

ctr_restrictions(
  cutset,
  ['SIZE_CUTSET'>=0,
   'SIZE_CUTSET'=<size('NODES'),
   required('NODES', [index, succ, bool]),
   'NODES'^index>=1,
   'NODES'^index=<size('NODES'),
   distinct('NODES', index),
   'NODES'^bool>=0,
   'NODES'^bool=<1]).

ctr_graph(
  cutset,
  ['NODES'],
  2,
  ['CLIQUE'>>collection(nodes1, nodes2)],
  [in_set(nodes2^index, nodes1^succ),
   nodes1^bool=1,
   nodes2^bool=1],
  ['MAX_NSCC'=<1, 'NVERTEX'=size('NODES')-'SIZE_CUTSET']).

ctr_example(
  cutset,
  cutset(
    1,
    [[index-1, succ-{2, 3, 4}, bool-1],
     [index-2, succ-{3}, bool-1],
     [index-3, succ-{4}, bool-1],
     [index-4, succ-{1}, bool-0]])).

```

B.60 cycle

```

ctr_date(cycle, ['20000128', '20030820']).

ctr_origin(cycle, '\\cite{BeldiceanuContejean94}', []).

ctr_arguments(
    cycle,
    ['NCYCLE'-dvar, 'NODES'-collection(index-int, succ-dvar)]).

ctr_restrictions(
    cycle,
    ['NCYCLE'>=1,
     'NCYCLE'=<size('NODES'),
     required('NODES', [index, succ]),
     'NODES'^index>=1,
     'NODES'^index=<size('NODES'),
     distinct('NODES', index),
     'NODES'^succ>=1,
     'NODES'^succ=<size('NODES')]).

ctr_graph(
    cycle,
    ['NODES'],
    2,
    ['CLIQUE'>>collection(nodes1, nodes2)],
    [nodes1^succ=nodes2^index],
    ['NTREE'=0, 'NCC'='NCYCLE']).

ctr_example(
    cycle,
    cycle(
        2,
        [[index-1, succ-2],
         [index-2, succ-1],
         [index-3, succ-5],
         [index-4, succ-3],
         [index-5, succ-4]])).

```

B.61 cycle_card_on_path

```
ctr_date(cycle_card_on_path, ['20000128', '20030820', '20040530']).
```

```
ctr_origin(cycle_card_on_path, 'CHIP', []).
```

```
ctr_arguments(
  cycle_card_on_path,
  ['NCYCLE'-dvar,
   'NODES'-collection(index-int, succ-dvar, colour-dvar),
   'ATLEAST'-int,
   'ATMOST'-int,
   'PATH_LEN'-int,
   'VALUES'-collection(val-int)].
```

```
ctr_restrictions(
  cycle_card_on_path,
  ['NCYCLE'>=1,
   'NCYCLE'=<size('NODES'),
   required('NODES', [index, succ, colour]),
   'NODES'^index>=1,
   'NODES'^index<size('NODES'),
   distinct('NODES', index),
   'NODES'^succ>=1,
   'NODES'^succ<size('NODES'),
   'ATLEAST'>=0,
   'ATLEAST'=<'PATH_LEN',
   'ATMOST'>='ATLEAST',
   'PATH_LEN'>=0,
   required('VALUES', val),
   distinct('VALUES', val)].
```

```
ctr_graph(
  cycle_card_on_path,
  ['NODES'],
  2,
  ['CLIQUE'>>collection(nodes1, nodes2)],
  [nodes1^succ=nodes2^index],
  ['NTREE'=0, 'NCC'='NCYCLE'],
  [>>('PATH_LENGTH'('PATH_LEN'),
    [- (variables,
        col('VARIABLES'-collection(var-dvar),
            [item(var-'NODES'^colour)]))]),
    [among_low_up('ATLEAST', 'ATMOST', variables, 'VALUES')]).
```

```
ctr_example(
```



```
cycle_card_on_path,  
cycle_card_on_path(  
  2,  
  [[index-1,succ-7,colour-2],  
   [index-2,succ-4,colour-3],  
   [index-3,succ-8,colour-2],  
   [index-4,succ-9,colour-1],  
   [index-5,succ-1,colour-2],  
   [index-6,succ-2,colour-1],  
   [index-7,succ-5,colour-1],  
   [index-8,succ-6,colour-1],  
   [index-9,succ-3,colour-1]],  
  1,  
  2,  
  3,  
  [[val-1]]).  
.
```

B.62 cycle_or_accessibility

```

ctr_date(cycle_or_accessibility, ['20000128', '20030820']).

ctr_origin(
    cycle_or_accessibility,
    'Inspired by \\cite{LabbeLaporteRodriguezMartin98}.',
    []).

ctr_arguments(
    cycle_or_accessibility,
    ['MAXDIST'-int,
     'NCYCLE'-dvar,
     'NODES'-collection(index-int, succ-dvar, x-int, y-int)]).

ctr_restrictions(
    cycle_or_accessibility,
    ['MAXDIST'>=0,
     'NCYCLE'>=1,
     'NCYCLE'=<size('NODES'),
     required('NODES', [index, succ, x, y]),
     'NODES'^index>=1,
     'NODES'^index=<size('NODES'),
     distinct('NODES', index),
     'NODES'^succ>=0,
     'NODES'^succ=<size('NODES'),
     'NODES'^x>=0,
     'NODES'^y>=0]).

ctr_graph(
    cycle_or_accessibility,
    ['NODES'],
    2,
    ['CLIQUE'>>collection(nodes1, nodes2)],
    [nodes1^succ=nodes2^index],
    ['NTREE'=0, 'NCC'='NCYCLE']).

ctr_graph(
    cycle_or_accessibility,
    ['NODES'],
    2,
    ['CLIQUE'>>collection(nodes1, nodes2)],
    [#\/(nodes1^succ=nodes2^index,
     #/\(nodes1^succ=0#\nodes2^succ=\=0,
     =<(+(abs(nodes1^x-nodes2^x),
          abs(nodes1^y-nodes2^y))),

```

```

        'MAXDIST' ))]],
['NVERTEX'=size('NODES')],
[>>('PRED',
    [-(variables,
        col('VARIABLES'-collection(var-dvar),
            [item(var-'NODES'^succ)])),
        destination]],
[nvalues_except_0(variables,=,1)]).

ctr_example(
    cycle_or_accessibility,
    cycle_or_accessibility(
        3,
        2,
        [[index-1,succ-6,x-4,y-5],
        [index-2,succ-0,x-9,y-1],
        [index-3,succ-0,x-2,y-4],
        [index-4,succ-1,x-2,y-6],
        [index-5,succ-5,x-7,y-2],
        [index-6,succ-4,x-4,y-7],
        [index-7,succ-0,x-6,y-4]])).

```

B.63 cycle_resource

```
ctr_date(cycle_resource, ['20030820', '20040530']).
```

```
ctr_origin(cycle_resource, 'CHIP', []).
```

```
ctr_arguments(
  cycle_resource,
  [-( 'RESOURCE',
      collection(id-int, first_task-dvar, nb_task-dvar)),
    'TASK'-collection(id-int, next_task-dvar, resource-dvar)]).
```

```
ctr_restrictions(
  cycle_resource,
  [required('RESOURCE', [id, first_task, nb_task]),
    'RESOURCE'^id>=1,
    'RESOURCE'^id<size('RESOURCE'),
    distinct('RESOURCE', id),
    'RESOURCE'^first_task>=1,
    'RESOURCE'^first_task<size('RESOURCE')+size('TASK'),
    'RESOURCE'^nb_task>=0,
    'RESOURCE'^nb_task<size('TASK'),
    required('TASK', [id, next_task, resource]),
    'TASK'^id>size('RESOURCE'),
    'TASK'^id<size('RESOURCE')+size('TASK'),
    distinct('TASK', id),
    'TASK'^next_task>=1,
    'TASK'^next_task<size('RESOURCE')+size('TASK'),
    'TASK'^resource>=1,
    'TASK'^resource<size('RESOURCE')]).
```

```
ctr_derived_collections(
  cycle_resource,
  [col(-( 'RESOURCE_TASK',
          collection(index-int, succ-dvar, name-dvar)),
    [item(
      index-'RESOURCE'^id,
      succ-'RESOURCE'^first_task,
      name-'RESOURCE'^id),
    item(
      index-'TASK'^id,
      succ-'TASK'^next_task,
      name-'TASK'^resource)])]).
```

```
ctr_graph(
  cycle_resource,
```

```

['RESOURCE_TASK'],
2,
['CLIQUE'>>collection(resource_task1,resource_task2)],
[resource_task1^succ=resource_task2^index,
resource_task1^name=resource_task2^name],
['NTREE'=0,
'NCC'=size('RESOURCE'),
'NVERTEX'=size('RESOURCE')+size('TASK')]).

ctr_graph(
cycle_resource,
['RESOURCE_TASK'],
2,
foreach(
'RESOURCE',
['CLIQUE'>>collection(resource_task1,resource_task2)]),
[resource_task1^succ=resource_task2^index,
resource_task1^name=resource_task2^name,
resource_task1^name='RESOURCE'^id],
['NVERTEX'='RESOURCE'^nb_task+1]).

ctr_example(
cycle_resource,
cycle_resource(
[[id-1,first_task-5,nb_task-3],
[id-2,first_task-2,nb_task-0],
[id-3,first_task-8,nb_task-2]],
[[id-4,next_task-7,resource-1],
[id-5,next_task-4,resource-1],
[id-6,next_task-3,resource-3],
[id-7,next_task-1,resource-1],
[id-8,next_task-6,resource-3]]).

```

B.64 cyclic_change

```

ctr_automaton(cyclic_change,cyclic_change).

ctr_date(cyclic_change,['20000128','20030820','20040530']).

ctr_origin(cyclic_change,'Derived from %c.',[change]).

ctr_arguments(
    cyclic_change,
    ['NCHANGE'-dvar,
     'CYCLE_LENGTH'-int,
     'VARIABLES'-collection(var-dvar),
     'CTR'-atom]).

ctr_restrictions(
    cyclic_change,
    ['NCHANGE'>=0,
     'NCHANGE'<size('VARIABLES'),
     'CYCLE_LENGTH'>0,
     required('VARIABLES',var),
     in_list('CTR',[=,\=,<,>=>,=<])]).

ctr_graph(
    cyclic_change,
    ['VARIABLES'],
    2,
    ['PATH'>>collection(variables1,variables2)],
    ['CTR'(
        (variables1^var+1)mod 'CYCLE_LENGTH',
        variables2^var)],
    ['NARC'='NCHANGE']).

ctr_example(
    cyclic_change,
    cyclic_change(
        2,
        4,
        [[var-3],[var-0],[var-2],[var-3],[var-1]],
        =\=)).

cyclic_change(A,B,C,D) :-
    cyclic_change_signature(C,E,D),
    automaton(
        E,
        F,

```

```

    E,
    0..1,
    [source(s), sink(t)],
    [arc(s, 0, s), arc(s, 1, s, [G+1]), arc(s, $, t)],
    [G],
    [0],
    [A]).

cyclic_change_signature([], [], A).

cyclic_change_signature([A], [], B) :-
    !.

cyclic_change_signature([[var-A], [var-B] | C], [D|E], =) :-
    !,
    (A+1) mod F# = B# <=> D,
    cyclic_change_signature([[var-B] | C], E, =).

cyclic_change_signature([[var-A], [var-B] | C], [D|E], =\=) :-
    !,
    (A+1) mod F#\= B# <=> D,
    cyclic_change_signature([[var-B] | C], E, =\=).

cyclic_change_signature([[var-A], [var-B] | C], [D|E], <) :-
    !,
    (A+1) mod F# < B# <=> D,
    cyclic_change_signature([[var-B] | C], E, <).

cyclic_change_signature([[var-A], [var-B] | C], [D|E], >=) :-
    !,
    (A+1) mod F# >= B# <=> D,
    cyclic_change_signature([[var-B] | C], E, >=).

cyclic_change_signature([[var-A], [var-B] | C], [D|E], >) :-
    !,
    (A+1) mod F# > B# <=> D,
    cyclic_change_signature([[var-B] | C], E, >).

cyclic_change_signature([[var-A], [var-B] | C], [D|E], =<) :-
    !,
    (A+1) mod F# =< B# <=> D,
    cyclic_change_signature([[var-B] | C], E, =<).

```

B.65 cyclic_change_joker

```
ctr_automaton(cyclic_change_joker,cyclic_change_joker).
```

```
ctr_date(
    cyclic_change_joker,
    ['20000128','20030820','20040530']).
```

```
ctr_origin(
    cyclic_change_joker,
    'Derived from %c.',
    [cyclic_change]).
```

```
ctr_arguments(
    cyclic_change_joker,
    ['NCHANGE'-dvar,
     'CYCLE_LENGTH'-int,
     'VARIABLES'-collection(var-dvar),
     'CTR'-atom]).
```

```
ctr_restrictions(
    cyclic_change_joker,
    ['NCHANGE'>=0,
     'NCHANGE'<size('VARIABLES'),
     required('VARIABLES',var),
     'CYCLE_LENGTH'>0,
     in_list('CTR',[=,\=,<,>=>,>=<])]).
```

```
ctr_graph(
    cyclic_change_joker,
    ['VARIABLES'],
    2,
    ['PATH'>>collection(variables1,variables2)],
    ['CTR'(
        (variables1^var+1)mod 'CYCLE_LENGTH',
        variables2^var),
     variables1^var<'CYCLE_LENGTH',
     variables2^var<'CYCLE_LENGTH'],
    ['NARC'='NCHANGE']).
```

```
ctr_example(
    cyclic_change_joker,
    cyclic_change_joker(
        2,
        4,
        [[var-3],
```



```

[var-0],
[var-2],
[var-4],
[var-4],
[var-4],
[var-3],
[var-1],
[var-4]],
=\=) .

```

```

cyclic_change_joker(A,B,C,D) :-
  cyclic_change_joker_signature(C,E,B,D),
  automaton(
    E,
    F,
    E,
    0..1,
    [source(s),sink(t)],
    [arc(s,0,s),arc(s,1,s,[G+1]),arc(s,$,t)],
    [G],
    [0],
    [A]) .

```

```

cyclic_change_joker_signature([],[],A,B) .

```

```

cyclic_change_joker_signature([A],[],B,C) :-
  ! .

```

```

cyclic_change_joker_signature([[var-A],[var-B]|C],[D|E],F,=) :-
  !,
  (A+1) mod F# = B# / \A# < F# / \B# < F# < => D,
  cyclic_change_joker_signature([[var-B]|C],E,F,=) .

```

```

cyclic_change_joker_signature([[var-A],[var-B]|C],[D|E],F,=\=) :-
  !,
  (A+1) mod F# \= B# / \A# < F# / \B# < F# < => D,
  cyclic_change_joker_signature([[var-B]|C],E,F,=\=) .

```

```

cyclic_change_joker_signature([[var-A],[var-B]|C],[D|E],F,<) :-
  !,
  (A+1) mod F# < B# / \A# < F# / \B# < F# < => D,
  cyclic_change_joker_signature([[var-B]|C],E,F,<) .

```

```

cyclic_change_joker_signature([[var-A],[var-B]|C],[D|E],F,>=) :-
  !,
  (A+1) mod F# >= B# / \A# < F# / \B# < F# < => D,

```

```

cyclic_change_joker_signature([ [var-B] | C], E, F, >=) .

cyclic_change_joker_signature([ [var-A], [var-B] | C], [D|E], F, >) :-
    !,
    (A+1) mod F #> B #/\ A #< F #/\ B #< F #<=> D,
    cyclic_change_joker_signature([ [var-B] | C], E, F, >) .

cyclic_change_joker_signature([ [var-A], [var-B] | C], [D|E], F, =<) :-
    !,
    (A+1) mod F #=< B #/\ A #< F #/\ B #< F #<=> D,
    cyclic_change_joker_signature([ [var-B] | C], E, F, =<) .

```

B.66 decreasing

```

ctr_automaton(decreasing,decreasing) .

ctr_date(decreasing,['20040814']) .

ctr_origin(decreasing,'Inspired by %c.',[increasing]) .

ctr_arguments(decreasing,['VARIABLES'-collection(var-dvar)]) .

ctr_restrictions(
    decreasing,
    [size('VARIABLES')>0,required('VARIABLES',var)]) .

ctr_graph(
    decreasing,
    ['VARIABLES'],
    2,
    ['PATH'>>collection(variables1,variables2)],
    [variables1^var>=variables2^var],
    ['NARC'=size('VARIABLES')-1]) .

ctr_example(
    decreasing,
    decreasing([[var-8],[var-4],[var-1],[var-1]])) .

decreasing(A) :-
    decreasing_signature(A,B),
    automaton(
        B,
        C,
        B,
        0..1,
        [source(s),sink(t)],
        [arc(s,0,s),arc(s,$,t)],
        [],
        [],
        []).

decreasing_signature([A],[ ]).

decreasing_signature([[var-A],[var-B]|C],[D|E]) :-
    in(D,0..1),
    A#<B#<=>D,
    decreasing_signature([[var-B]|C],E) .

```

B.67 deepest_valley

```

ctr_automaton(deepest_valley,deepest_valley).

ctr_date(deepest_valley,['20040530']).

ctr_origin(deepest_valley,'Derived from %c.',[valley]).

ctr_arguments(
    deepest_valley,
    ['DEPTH'-dvar,'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    deepest_valley,
    ['DEPTH'>=0,'VARIABLES'^var>=0,required('VARIABLES',var)]).

ctr_example(
    deepest_valley,
    deepest_valley(
        2,
        [[var-5],
         [var-3],
         [var-4],
         [var-8],
         [var-8],
         [var-2],
         [var-7],
         [var-1]])).

deepest_valley(A,B) :-
    C=1000000,
    deepest_valley_signature(B,D,E),
    automaton(
        E,
        F-G,
        D,
        0..2,
        [source(s),node(u),sink(t)],
        [arc(s,0,s),
         arc(s,1,s),
         arc(s,2,u),
         arc(s,$,t),
         arc(u,0,s,[min(H,F)]),
         arc(u,1,u),
         arc(u,2,u),
         arc(u,$,t)],

```

```
[H],  
[C],  
[A]).
```

```
deepest_valley_signature([], [], []).
```

```
deepest_valley_signature([A], [], []).
```

```
deepest_valley_signature([[var-A], [var-B] | C], [D | E], [A-B | F]) :-  
    in(D, 0..2),  
    A#<B#<=>D#=0,  
    A#=B#<=>D#=1,  
    A#>B#<=>D#=2,  
    deepest_valley_signature([[var-B] | C], E, F).
```

B.68 derangement

```

ctr_date(derangement, ['20000128', '20030820', '20040530']).

ctr_origin(derangement, 'Derived from %c.', [cycle]).

ctr_arguments(
    derangement,
    ['NODES'-collection(index-int, succ-dvar)]).

ctr_restrictions(
    derangement,
    [required('NODES', [index, succ]),
     'NODES'^index>=1,
     'NODES'^index<=size('NODES'),
     distinct('NODES', index),
     'NODES'^succ>=1,
     'NODES'^succ<=size('NODES')]).

ctr_graph(
    derangement,
    ['NODES'],
    2,
    ['CLIQUE'>>collection(nodes1, nodes2)],
    [nodes1^succ=nodes2^index, nodes1^succ=\=nodes1^index],
    ['NTREE'=0]).

ctr_example(
    derangement,
    derangement(
        [[index-1, succ-2],
         [index-2, succ-1],
         [index-3, succ-5],
         [index-4, succ-3],
         [index-5, succ-4]])).

```

B.69 differ_from_at_least_k_pos

```

ctr_automaton(
    differ_from_at_least_k_pos,
    differ_from_at_least_k_pos).

ctr_date(differ_from_at_least_k_pos, ['20030820', '20040530']).

ctr_origin(
    differ_from_at_least_k_pos,
    'Inspired by \\cite{Frutos97}.',
    []).

ctr_types(
    differ_from_at_least_k_pos,
    ['VECTOR'-collection(var-dvar)]).

ctr_arguments(
    differ_from_at_least_k_pos,
    ['K'-int, 'VECTOR1'-'VECTOR', 'VECTOR2'-'VECTOR']).

ctr_restrictions(
    differ_from_at_least_k_pos,
    [required('VECTOR', var),
     'K'>=0,
     'K'=<size('VECTOR1'),
     size('VECTOR1')=size('VECTOR2')]).

ctr_graph(
    differ_from_at_least_k_pos,
    ['VECTOR1', 'VECTOR2'],
    2,
    ['PRODUCT' (=)>>collection(vector1, vector2)],
    [vector1^var=\=vector2^var],
    ['NARC'>='K']).

ctr_example(
    differ_from_at_least_k_pos,
    differ_from_at_least_k_pos(
        2,
        [[var-2], [var-5], [var-2], [var-0]],
        [[var-3], [var-6], [var-2], [var-1]])).

differ_from_at_least_k_pos(A, B, C) :-
    differ_from_at_least_k_pos_signature(B, C, D),
    E#>=A,

```

```
automaton(  
    D,  
    F,  
    D,  
    0..1,  
    [source(s),sink(t)],  
    [arc(s,0,s,[G+1]),arc(s,1,s),arc(s,$,t)],  
    [G],  
    [0],  
    [E]).  
  
differ_from_at_least_k_pos_signature([],[],[]).  
  
differ_from_at_least_k_pos_signature(  
    [[var-A]|B],  
    [[var-C]|D],  
    [E|F]) :-  
    A#=C#<=>E,  
    differ_from_at_least_k_pos_signature(B,D,F).
```


B.70 diffn

```

ctr_date(diffn, ['20000128', '20030820', '20040530']).

ctr_origin(diffn, '\\cite{BeldiceanuContejean94}', []).

ctr_types(
    diffn,
    ['ORTHOTOPE'-collection(ori-dvar, siz-dvar, end-dvar)]).

ctr_arguments(
    diffn,
    ['ORTHOPESES'-collection(orth-'ORTHOTOPE')]).

ctr_restrictions(
    diffn,
    [size('ORTHOTOPE')>0,
     require_at_least(2, 'ORTHOTOPE', [ori, siz, end]),
     'ORTHOTOPE'^siz>=0,
     required('ORTHOPESES', orth),
     same_size('ORTHOPESES', orth)]).

ctr_graph(
    diffn,
    ['ORTHOPESES'],
    1,
    ['SELF'>>collection(orthotopes)],
    [orth_link_ori_siz_end(orthotopes^orth)],
    ['NARC'=size('ORTHOPESES')]).

ctr_graph(
    diffn,
    ['ORTHOPESES'],
    2,
    ['CLIQUE' (=\\=)>>collection(orthotopes1, orthotopes2)],
    [two_orth_do_not_overlap(
        orthotopes1^orth,
        orthotopes2^orth)],
    [=('NARC',
        -(size('ORTHOPESES')*size('ORTHOPESES'),
        size('ORTHOPESES')))]).

ctr_example(
    diffn,
    diffn(
        [[orth-[[ori-2, siz-2, end-4], [ori-1, siz-3, end-4]]],

```

```
[orth-[[ori-4,siz-4,end-8],[ori-3,siz-3,end-3]],  
[orth-[[ori-9,siz-2,end-11],[ori-4,siz-3,end-7]]]]) .
```

B.71 diffn_column

```

ctr_date(diffn_column, ['20030820']).

ctr_origin(
    diffn_column,
    'CHIP: option guillotine cut (column) of %c.',
    [diffn]).

ctr_types(
    diffn_column,
    ['ORTHOTOPE'-collection(ori-dvar, siz-dvar, end-dvar)]).

ctr_arguments(
    diffn_column,
    ['ORTHOTOPES'-collection(orth-'ORTHOTOPE'), 'N'-int]).

ctr_restrictions(
    diffn_column,
    [size('ORTHOTOPE')>0,
     require_at_least(2, 'ORTHOTOPE', [ori, siz, end]),
     'ORTHOTOPE'^siz>=0,
     required('ORTHOTOPES', orth),
     same_size('ORTHOTOPES', orth),
     'N'>0,
     'N'=<size('ORTHOTOPE'),
     diffn('ORTHOTOPES')]).

ctr_graph(
    diffn_column,
    ['ORTHOTOPES'],
    2,
    ['CLIQUE' (<)>>collection(orthotopes1, orthotopes2)],
    [two_orth_column(orthotopes1^orth, orthotopes2^orth, 'N')],
    ['NARC'=size('ORTHOTOPES')*(size('ORTHOTOPES')-1)/2]).

ctr_example(
    diffn_column,
    diffn_column(
        [[orth-[[ori-1, siz-3, end-4], [ori-1, siz-1, end-2]]],
         [orth-[[ori-4, siz-2, end-6], [ori-1, siz-3, end-4]]]],
        1)).

```

B.72 `diffn_include`

```

ctr_date(diffn_include, ['20030820']).

ctr_origin(
    diffn_include,
    'CHIP: option guillotine cut (include) of %c.',
    [diffn]).

ctr_types(
    diffn_include,
    ['ORTHOTOPE'-collection(ori-dvar, siz-dvar, end-dvar)]).

ctr_arguments(
    diffn_include,
    ['ORTHOTOPES'-collection(orth-'ORTHOTOPE'), 'N'-int]).

ctr_restrictions(
    diffn_include,
    [size('ORTHOTOPE')>0,
     require_at_least(2, 'ORTHOTOPE', [ori, siz, end]),
     'ORTHOTOPE'^siz>=0,
     required('ORTHOTOPES', orth),
     same_size('ORTHOTOPES', orth),
     'N'>0,
     'N'<=size('ORTHOTOPE'),
     diffn('ORTHOTOPES')]).

ctr_graph(
    diffn_include,
    ['ORTHOTOPES'],
    2,
    ['CLIQUE' (<)>>collection(orthotopes1, orthotopes2)],
    [two_orth_include(orthotopes1^orth, orthotopes2^orth, 'N')],
    ['NARC'=size('ORTHOTOPES')*(size('ORTHOTOPES')-1)/2]).

ctr_example(
    diffn_include,
    diffn_include(
        [[orth-[[ori-1, siz-3, end-4], [ori-1, siz-1, end-2]]],
         [orth-[[ori-1, siz-2, end-3], [ori-2, siz-3, end-5]]]],
        1)).

```

B.73 discrepancy

```

ctr_date(discrepancy, ['20050506']).

ctr_origin(
  discrepancy,
  '\\cite{Focacci01} and \\cite{vanHoeve05}',
  []).

ctr_arguments(
  discrepancy,
  ['VARIABLES'-collection(var-dvar,bad-sint),'K'-int]).

ctr_restrictions(
  discrepancy,
  [required('VARIABLES',var),
   required('VARIABLES',bad),
   'K'>=0,
   'K'=<size('VARIABLES')]).

ctr_graph(
  discrepancy,
  ['VARIABLES'],
  1,
  ['SELF'>>collection(variables)],
  [in_set(variables^var,variables^bad)],
  ['NARC'='K']).

ctr_example(
  discrepancy,
  discrepancy(
    [[var-4,bad-{1,4,6}],
     [var-5,bad-{0,1}],
     [var-5,bad-{1,6,9}],
     [var-4,bad-{1,4}],
     [var-1,bad-{}]],
    2)).

```

B.74 disjoint

```

ctr_date(disjoint,['20000315','20031017','20040530']).

ctr_origin(disjoint,'Derived from %c.',[alldifferent]).

ctr_arguments(
    disjoint,
    ['VARIABLES1'-collection(var-dvar),
     'VARIABLES2'-collection(var-dvar)]).

ctr_restrictions(
    disjoint,
    [required('VARIABLES1',var),required('VARIABLES2',var)]).

ctr_graph(
    disjoint,
    ['VARIABLES1','VARIABLES2'],
    2,
    ['PRODUCT'>>collection(variables1,variables2)],
    [variables1^var=variables2^var],
    ['NARC'=0]).

ctr_example(
    disjoint,
    disjoint(
        [[var-1],[var-9],[var-1],[var-5]],
        [[var-2],[var-7],[var-7],[var-0],[var-6],[var-8]])).

```

B.75 disjoint_tasks

```

ctr_date(disjoint_tasks, ['20030820']).

ctr_origin(disjoint_tasks, 'Derived from %c.', [disjoint]).

ctr_arguments(
    disjoint_tasks,
    ['TASKS1'-collection(origin-dvar,duration-dvar,end-dvar),
     'TASKS2'-collection(origin-dvar,duration-dvar,end-dvar)]).

ctr_restrictions(
    disjoint_tasks,
    [require_at_least(2, 'TASKS1', [origin,duration,end]),
     'TASKS1'^duration>=0,
     require_at_least(2, 'TASKS2', [origin,duration,end]),
     'TASKS2'^duration>=0]).

ctr_graph(
    disjoint_tasks,
    ['TASKS1'],
    1,
    ['SELF'>>collection(tasks1)],
    [tasks1^origin+tasks1^duration=tasks1^end],
    ['NARC'=size('TASKS1')]).

ctr_graph(
    disjoint_tasks,
    ['TASKS2'],
    1,
    ['SELF'>>collection(tasks2)],
    [tasks2^origin+tasks2^duration=tasks2^end],
    ['NARC'=size('TASKS2')]).

ctr_graph(
    disjoint_tasks,
    ['TASKS1', 'TASKS2'],
    2,
    ['PRODUCT'>>collection(tasks1,tasks2)],
    [tasks1^duration>0,
     tasks2^duration>0,
     tasks1^origin<tasks2^end,
     tasks2^origin<tasks1^end],
    ['NARC'=0]).

ctr_example(

```

```
disjoint_tasks,  
disjoint_tasks(  
    [[origin-6,duration-5,end-11],  
     [origin-8,duration-2,end-10]],  
    [[origin-2,duration-2,end-4],  
     [origin-3,duration-3,end-6],  
     [origin-12,duration-1,end-13]])).
```


B.76 disjunctive

```

ctr_date(disjunctive, ['20050506']).

ctr_origin(disjunctive, '\\cite{Carrier82}', []).

ctr_synonyms(disjunctive, [one_machine]).

ctr_arguments(
    disjunctive,
    ['TASKS'-collection(origin-dvar, duration-dvar)]).

ctr_restrictions(
    disjunctive,
    [required('TASKS', [origin, duration]), 'TASKS'^duration>=0]).

ctr_graph(
    disjunctive,
    ['TASKS'],
    2,
    ['CLIQUE' (<)>>collection(tasks1, tasks2)],
    [#\ / (\#\ / (tasks1^duration=0#\ / tasks2^duration=0,
        tasks1^origin+tasks1^duration=<tasks2^origin),
        tasks2^origin+tasks2^duration=<tasks1^origin)],
    ['NARC'=size('TASKS')*(size('TASKS')-1)/2]).

ctr_example(
    disjunctive,
    disjunctive(
        [[origin-1, duration-3],
         [origin-2, duration-0],
         [origin-7, duration-2],
         [origin-4, duration-1]])).

```

B.77 distance_between

```

ctr_date(distance_between, ['20000128', '20030820']).

ctr_origin(distance_between, 'N.~Beldiceanu', []).

ctr_arguments(
  distance_between,
  ['DIST'-dvar,
   'VARIABLES1'-collection(var-dvar),
   'VARIABLES2'-collection(var-dvar),
   'CTR'-atom]).

ctr_restrictions(
  distance_between,
  ['DIST'>=0,
   'DIST'=<size('VARIABLES1')*size('VARIABLES2'),
   required('VARIABLES1',var),
   required('VARIABLES2',var),
   size('VARIABLES1')=size('VARIABLES2'),
   in_list('CTR', [=,\=,<,>=,>,<=])]).

ctr_graph(
  distance_between,
  [['VARIABLES1'], ['VARIABLES2']],
  2,
  ['CLIQUE' (=\\=)>>collection(variables1,variables2)],
  ['CTR' (variables1^var,variables2^var)],
  ['DISTANCE'='DIST']).

ctr_example(
  distance_between,
  distance_between(
    2,
    [[var-3],[var-4],[var-6],[var-2],[var-4]],
    [[var-2],[var-6],[var-9],[var-3],[var-6]],
    <)).

```

B.78 distance_change

```

ctr_automaton(distance_change,distance_change).

ctr_date(distance_change,['20000128','20030820','20040530']).

ctr_origin(distance_change,'Derived from %c.',[change]).

ctr_arguments(
    distance_change,
    ['DIST'-dvar,
     'VARIABLES1'-collection(var-dvar),
     'VARIABLES2'-collection(var-dvar),
     'CTR'-atom]).

ctr_restrictions(
    distance_change,
    ['DIST'>=0,
     'DIST'<size('VARIABLES1'),
     required('VARIABLES1',var),
     required('VARIABLES2',var),
     size('VARIABLES1')=size('VARIABLES2'),
     in_list('CTR',[=,\=,<,>=>,=<])]).

ctr_graph(
    distance_change,
    [['VARIABLES1'],['VARIABLES2']],
    2,
    ['PATH'>>collection(variables1,variables2)],
    ['CTR'(variables1^var,variables2^var)],
    ['DISTANCE'='DIST']).

ctr_example(
    distance_change,
    distance_change(
        1,
        [[var-3],[var-3],[var-1],[var-2],[var-2]],
        [[var-4],[var-4],[var-3],[var-3],[var-3]],
        =\=)).

distance_change(A,B,C,D) :-
    distance_change_signature(B,C,E,D),
    automaton(
        E,
        F,
        E,

```

```

0..1,
[source(s),sink(t)],
[arc(s,0,s),arc(s,1,s,[G+1]),arc(s,$,t)],
[G],
[0],
[A]).

```

```
distance_change_signature([],[],[],A).
```

```
distance_change_signature([A],[B],[],C) :-
!.
```

```
distance_change_signature(
[[var-A],[var-B]|C],
[[var-D],[var-E]|F],
[G|H],
=) :-
!,
A#=B#/\D#\=E#\A#=B#/\D#=E#<=>G,
distance_change_signature([[var-B]|C],[[var-E]|F],H,=).
```

```
distance_change_signature(
[[var-A],[var-B]|C],
[[var-D],[var-E]|F],
[G|H],
=\=) :-
!,
A#\=B#/\D#=E#\A#=B#/\D#\=E#<=>G,
distance_change_signature(
[[var-B]|C],
[[var-E]|F],
H,
=\=).
```

```
distance_change_signature(
[[var-A],[var-B]|C],
[[var-D],[var-E]|F],
[G|H],
<) :-
!,
A#<B#/\D#>=E#\A#>=B#/\D#<E#<=>G,
distance_change_signature([[var-B]|C],[[var-E]|F],H,<).
```

```
distance_change_signature(
[[var-A],[var-B]|C],
[[var-D],[var-E]|F],
```

```

[G|H],
>=) :-
    !,
    A#>=B#/\D#<E#\A#<B#/\D#>=E#<=>G,
    distance_change_signature([ [var-B] |C], [ [var-E] |F], H, >=) .

```

```

distance_change_signature(
    [ [var-A], [var-B] |C],
    [ [var-D], [var-E] |F],
    [G|H],
    >) :-
    !,
    A#>B#/\D#=<E#\A#=<B#/\D#>E#<=>G,
    distance_change_signature([ [var-B] |C], [ [var-E] |F], H, >) .

```

```

distance_change_signature(
    [ [var-A], [var-B] |C],
    [ [var-D], [var-E] |F],
    [G|H],
    =<) :-
    !,
    A#=<B#/\D#>E#\A#>B#/\D#=<E#<=>G,
    distance_change_signature([ [var-B] |C], [ [var-E] |F], H, =<) .

```

B.79 domain_constraint

```

ctr_automaton(domain_constraint, domain_constraint).

ctr_date(domain_constraint, ['20030820', '20040530']).

ctr_origin(domain_constraint, '\\cite{Refalo00}', []).

ctr_arguments(
    domain_constraint,
    ['VAR'-dvar, 'VALUES'-collection(var01-dvar, value-int)]).

ctr_restrictions(
    domain_constraint,
    [required('VALUES', [var01, value]),
     'VALUES'^var01>=0,
     'VALUES'^var01=<1,
     distinct('VALUES', value)]).

ctr_derived_collections(
    domain_constraint,
    [col('VALUE'-collection(var01-int, value-dvar),
        [item(var01-1, value-'VAR')])]).

ctr_graph(
    domain_constraint,
    ['VALUE', 'VALUES'],
    2,
    ['PRODUCT'>>collection(value, values)],
    [value^value=values^value#<=>values^var01=1],
    ['NARC'=size('VALUES')]).

ctr_example(
    domain_constraint,
    domain_constraint(
        5,
        [[var01-0, value-9],
         [var01-1, value-5],
         [var01-0, value-2],
         [var01-0, value-7]])).

domain_constraint(A, B) :-
    domain_constraint_signature(B, C, A),
    automaton(
        C,
        D,

```

```
C,  
0..1,  
[source(s),sink(t)],  
[arc(s,l,s),arc(s,$,t)],  
[],  
[],  
[]).
```

```
domain_constraint_signature([],[],A).
```

```
domain_constraint_signature([[var01-A,value-B]|C],[D|E],F) :-  
    F#=B#<=>A#<=>D,  
    domain_constraint_signature(C,E,F).
```

B.80 elem

```

ctr_automaton(elem,elem) .

ctr_date(elem, ['20030820', '20040530']) .

ctr_origin(elem, 'Derived from %c.', [element]) .

ctr_usual_name(elem, element) .

ctr_arguments(
    elem,
    ['ITEM'-collection(index-dvar, value-dvar),
     'TABLE'-collection(index-int, value-dvar)] .

ctr_restrictions(
    elem,
    [required('ITEM', [index, value]),
     'ITEM'^index>=1,
     'ITEM'^index<=size('TABLE'),
     size('ITEM')=1,
     required('TABLE', [index, value]),
     'TABLE'^index>=1,
     'TABLE'^index<=size('TABLE'),
     distinct('TABLE', index)] .

ctr_graph(
    elem,
    ['ITEM', 'TABLE'],
    2,
    ['PRODUCT'>>collection(item, table)],
    [item^index=table^index, item^value=table^value],
    ['NARC'=1]) .

ctr_example(
    elem,
    elem(
        [[index-3, value-2]],
        [[index-1, value-6],
         [index-2, value-9],
         [index-3, value-2],
         [index-4, value-9]]) .

elem(A, B) :-
    A=[[index-C, value-D]],
    elem_signature(B, E, C, D),

```



```
automaton(  
    E,  
    F,  
    E,  
    0..1,  
    [source(s),sink(t)],  
    [arc(s,0,s),arc(s,1,t)],  
    [],  
    [],  
    []).  
  
elem_signature([],[],A,B).  
  
elem_signature([[index-A,value-B]|C],[D|E],F,G) :-  
    F#=A#/\G#=B#<=>D,  
    elem_signature(C,E,F,G).
```

B.81 element

```

ctr_automaton(element,element_).

ctr_date(element,['20000128','20030820','20040530']).

ctr_origin(element,'\cite{VanHentenryckCarillon88}',[]).

ctr_arguments(
    element,
    ['INDEX'-dvar,'TABLE'-collection(value-dvar),'VALUE'-dvar]).

ctr_restrictions(
    element,
    ['INDEX'>=1,
     'INDEX'=<size('TABLE'),
     required('TABLE',value)]).

ctr_derived_collections(
    element,
    [col('ITEM'-collection(index-dvar,value-dvar),
        [item(index-'INDEX',value-'VALUE')])]).

ctr_graph(
    element,
    ['ITEM','TABLE'],
    2,
    ['PRODUCT'>>collection(item,table)],
    [item^index=table^key,item^value=table^value],
    ['NARC'=1]).

ctr_example(
    element,
    element(3,[value-6],[value-9],[value-2],[value-9],2)).

element_(A,B,C):-
    length(B,D),
    in(A,1..D),
    element_signature(B,A,C,1,E),
    automaton(
        E,
        F,
        E,
        0..1,
        [source(s),sink(t)],
        [arc(s,0,s),arc(s,1,t)],

```

```
[ ],  
[ ],  
[ ]).
```

```
element_signature([ ],A,B,C,[ ]).
```

```
element_signature([[value-A] | B],C,D,E,[F|G]) :-  
  C#=E#/\D#=A#<=>F,  
  H is E+1,  
  element_signature(B,C,D,H,G).
```

B.82 element_greaterreq

```

ctr_automaton(element_greaterreq,element_greaterreq) .

ctr_date(element_greaterreq,['20030820','20040530']) .

ctr_origin(
  element_greaterreq,
  '\\cite{OttossonThorsteinssonHooker99}',
  []).

ctr_arguments(
  element_greaterreq,
  ['ITEM'-collection(index-dvar,value-dvar),
   'TABLE'-collection(index-int,value-int)]).

ctr_restrictions(
  element_greaterreq,
  [required('ITEM',[index,value]),
   'ITEM'^index>=1,
   'ITEM'^index<=size('TABLE'),
   size('ITEM')=1,
   required('TABLE',[index,value]),
   'TABLE'^index>=1,
   'TABLE'^index<=size('TABLE'),
   distinct('TABLE',index)]).

ctr_graph(
  element_greaterreq,
  ['ITEM','TABLE'],
  2,
  ['PRODUCT'>>collection(item,table)],
  [item^index=table^index,item^value>=table^value],
  ['NARC'=1]).

ctr_example(
  element_greaterreq,
  element_greaterreq(
    [[index-1,value-8]],
    [[index-1,value-6],
     [index-2,value-9],
     [index-3,value-2],
     [index-4,value-9]])).

element_greaterreq(A,B) :-
  A=[[index-C,value-D]],

```

```
element_greatereq_signature(B,E,C,D),
automaton(
    E,
    F,
    E,
    0..1,
    [source(s),sink(t)],
    [arc(s,0,s),arc(s,1,t)],
    [],
    [],
    []).
```

```
element_greatereq_signature([],[],A,B).
```

```
element_greatereq_signature([[index-A,value-B]|C],[D|E],F,G) :-
    F#=A#/\G#>=B#<=>D,
    element_greatereq_signature(C,E,F,G).
```

B.83 element_lesseq

```

ctr_automaton(element_lesseq,element_lesseq).

ctr_date(element_lesseq,['20030820','20040530']).

ctr_origin(
  element_lesseq,
  '\\cite{OttossonThorsteinssonHooker99}',
  []).

ctr_arguments(
  element_lesseq,
  ['ITEM'-collection(index-dvar,value-dvar),
   'TABLE'-collection(index-int,value-int)]).

ctr_restrictions(
  element_lesseq,
  [required('ITEM',[index,value]),
   'ITEM'^index>=1,
   'ITEM'^index<=size('TABLE'),
   size('ITEM')=1,
   required('TABLE',[index,value]),
   'TABLE'^index>=1,
   'TABLE'^index<=size('TABLE'),
   distinct('TABLE',index)]).

ctr_graph(
  element_lesseq,
  ['ITEM','TABLE'],
  2,
  ['PRODUCT'>>collection(item,table)],
  [item^index=table^index,item^value<=table^value],
  ['NARC'=1]).

ctr_example(
  element_lesseq,
  element_lesseq(
    [[index-3,value-1]],
    [[index-1,value-6],
     [index-2,value-9],
     [index-3,value-2],
     [index-4,value-9]])).

element_lesseq(A,B) :-
  A=[[index-C,value-D]],

```

```
element_lesseq_signature(B,E,C,D),
automaton(
    E,
    F,
    E,
    0..1,
    [source(s),sink(t)],
    [arc(s,0,s),arc(s,1,t)],
    [],
    [],
    []).
```

```
element_lesseq_signature([],[],A,B).
```

```
element_lesseq_signature([[index-A,value-B]|C],[D|E],F,G) :-
    F#=A#/\G#=<B#<=>D,
    element_lesseq_signature(C,E,F,G).
```

B.84 element_matrix

```

ctr_automaton(element_matrix,element_matrix).

ctr_date(element_matrix,['20031101']).

ctr_origin(element_matrix,'CHIP',[ ]).

ctr_arguments(
    element_matrix,
    ['MAX_I'-int,
     'MAX_J'-int,
     'INDEX_I'-dvar,
     'INDEX_J'-dvar,
     'MATRIX'-collection(i-int,j-int,v-int),
     'VALUE'-dvar]).

ctr_restrictions(
    element_matrix,
    ['MAX_I'>=1,
     'MAX_J'>=1,
     'INDEX_I'>=1,
     'INDEX_I'=<'MAX_I',
     'INDEX_J'>=1,
     'INDEX_J'=<'MAX_J',
     required('MATRIX',[i,j,v]),
     increasing_seq('MATRIX',[i,j]),
     'MATRIX'^i>=1,
     'MATRIX'^i=<'MAX_I',
     'MATRIX'^j>=1,
     'MATRIX'^j=<'MAX_J',
     size('MATRIX')='MAX_I'*'MAX_J']).

ctr_derived_collections(
    element_matrix,
    [col(-('ITEM',
           collection(index_i-dvar,index_j-dvar,value-dvar)),
         [item(
             index_i-'INDEX_I',
             index_j-'INDEX_J',
             value-'VALUE')]))]).

ctr_graph(
    element_matrix,
    ['ITEM','MATRIX'],
    2,

```



```

['PRODUCT' >> collection(item, matrix)],
[item^index_i=matrix^i,
 item^index_j=matrix^j,
 item^value=matrix^v],
['NARC'=1]).

ctr_example(
  element_matrix,
  element_matrix(
    4,
    3,
    1,
    3,
    [[i-1, j-1, v-4],
     [i-1, j-2, v-1],
     [i-1, j-3, v-7],
     [i-2, j-1, v-1],
     [i-2, j-2, v-0],
     [i-2, j-3, v-8],
     [i-3, j-1, v-3],
     [i-3, j-2, v-2],
     [i-3, j-3, v-1],
     [i-4, j-1, v-0],
     [i-4, j-2, v-0],
     [i-4, j-3, v-6]],
    7)).

element_matrix(A,B,C,D,E,F) :-
  in(C,1..A),
  in(D,1..B),
  element_matrix_signature(E,C,D,F,G),
  automaton(
    G,
    H,
    G,
    0..1,
    [source(s),sink(t)],
    [arc(s,0,s),arc(s,1,t)],
    [],
    [],
    []).

element_matrix_signature([],A,B,C,[]).

element_matrix_signature([[i-A,j-B,v-C]|D],E,F,G,[H|I]) :-
  E#=A#/\F#=B#/\G#=C#<=>H,

```

`element_matrix_signature(D,E,F,G,I).`

B.85 element_sparse

```

ctr_automaton(element_sparse,element_sparse).

ctr_date(element_sparse,['20030820','20040530']).

ctr_origin(element_sparse,'CHIP',[]).

ctr_usual_name(element_sparse,element).

ctr_arguments(
  element_sparse,
  ['ITEM'-collection(index-dvar,value-dvar),
   'TABLE'-collection(index-int,value-int),
   'DEFAULT'-int]).

ctr_restrictions(
  element_sparse,
  [required('ITEM',[index,value]),
   'ITEM'^index>=1,
   size('ITEM')=1,
   required('TABLE',[index,value]),
   'TABLE'^index>=1,
   distinct('TABLE',index)]).

ctr_derived_collections(
  element_sparse,
  [col('DEF'-collection(index-int,value-int),
      [item(index-0,value-'DEFAULT')]),
   col('TABLE_DEF'-collection(index-dvar,value-dvar),
      [item(index-'TABLE'^index,value-'TABLE'^value),
       item(index-'DEF'^index,value-'DEF'^value)])]).

ctr_graph(
  element_sparse,
  ['ITEM','TABLE_DEF'],
  2,
  ['PRODUCT'>>collection(item,table_def)],
  [item^value=table_def^value,
   item^index=table_def^index#\table_def^index=0],
  ['NARC'>=1]).

ctr_example(
  element_sparse,
  element_sparse(
    [[index-2,value-5]],

```

```

[[index-1,value-6],
 [index-2,value-5],
 [index-4,value-2],
 [index-8,value-9]],
5)).

```

```

element_sparse(A,B,C) :-
  A=[[index-D,value-E]],
  element_sparse_signature(B,F,D,E,C),
  automaton(
    F,
    G,
    F,
    0..2,
    [source(s),node(d),sink(t)],
    [arc(s,0,s),
     arc(s,1,t),
     arc(s,2,d),
     arc(d,1,t),
     arc(d,2,d),
     arc(d,$,t)],
    [],
    [],
    []).

```

```

element_sparse_signature([],[],A,B,C).

```

```

element_sparse_signature([[index-A,value-B]|C],[D|E],F,G,H) :-
  in(D,0..2),
  F#\=A#\G#\=H#\=<=>D#=0,
  F#=A#\G#=B#\=<=>D#=1,
  F#\=A#\G#=H#\=<=>D#=2,
  element_sparse_signature(C,E,F,G,H).

```

B.86 elements

```

ctr_date(elements, ['20030820']).

ctr_origin(elements, 'Derived from %c.', [element]).

ctr_arguments(
    elements,
    ['ITEMS'-collection(index-dvar, value-dvar),
     'TABLE'-collection(index-int, value-dvar)]).

ctr_restrictions(
    elements,
    [required('ITEMS', [index, value]),
     'ITEMS'^index>=1,
     'ITEMS'^index<=size('TABLE'),
     required('TABLE', [index, value]),
     'TABLE'^index>=1,
     'TABLE'^index<=size('TABLE'),
     distinct('TABLE', index)]).

ctr_graph(
    elements,
    ['ITEMS', 'TABLE'],
    2,
    ['PRODUCT'>>collection(items, table)],
    [items^index=table^index, items^value=table^value],
    ['NARC'=size('ITEMS')]).

ctr_example(
    elements,
    elements(
        [[index-4, value-9], [index-1, value-6]],
        [[index-1, value-6],
         [index-2, value-9],
         [index-3, value-2],
         [index-4, value-9]])).

```

B.87 elements_alldifferent

```

ctr_date(elements_alldifferent, ['20030820']).

ctr_origin(
  elements_alldifferent,
  'Derived from %c and %c.',
  [elements, alldifferent]).

ctr_synonyms(
  elements_alldifferent,
  [elements_alldiff, elements_alldistinct]).

ctr_arguments(
  elements_alldifferent,
  ['ITEMS'-collection(index-dvar, value-dvar),
   'TABLE'-collection(index-int, value-dvar)]).

ctr_restrictions(
  elements_alldifferent,
  [required('ITEMS', [index, value]),
   'ITEMS'^index>=1,
   'ITEMS'^index<=size('TABLE'),
   size('ITEMS')=size('TABLE'),
   required('TABLE', [index, value]),
   'TABLE'^index>=1,
   'TABLE'^index<=size('TABLE'),
   distinct('TABLE', index)]).

ctr_graph(
  elements_alldifferent,
  ['ITEMS', 'TABLE'],
  2,
  ['PRODUCT'>>collection(items, table)],
  [items^index=table^index, items^value=table^value],
  ['NVERTEX'=size('ITEMS')+size('TABLE')]).

ctr_example(
  elements_alldifferent,
  elements_alldifferent(
    [[index-2, value-9],
     [index-1, value-6],
     [index-4, value-9],
     [index-3, value-2]],
    [[index-1, value-6],
     [index-2, value-9],

```

[index-3,value-2],
[index-4,value-9]))).

B.88 elements_sparse

```

ctr_date(elements_sparse, ['20030820']).

ctr_origin(elements_sparse, 'Derived from %c.', [element_sparse]).

ctr_arguments(
    elements_sparse,
    ['ITEMS'-collection(index-dvar, value-dvar),
     'TABLE'-collection(index-int, value-int),
     'DEFAULT'-int]).

ctr_restrictions(
    elements_sparse,
    [required('ITEMS', [index, value]),
     'ITEMS'^index>=1,
     required('TABLE', [index, value]),
     'TABLE'^index>=1,
     distinct('TABLE', index)]).

ctr_derived_collections(
    elements_sparse,
    [col('DEF'-collection(index-int, value-int),
        [item(index-0, value-'DEFAULT')]),
     col('TABLE_DEF'-collection(index-dvar, value-dvar),
        [item(index-'TABLE'^index, value-'TABLE'^index),
         item(index-'DEF'^index, value-'DEF'^value)])]).

ctr_graph(
    elements_sparse,
    ['ITEMS', 'TABLE_DEF'],
    2,
    ['PRODUCT'>>collection(items, table_def)],
    [items^value=table_def^value,
     items^index=table_def^index#\table_def^index=0],
    ['NSOURCE'=size('ITEMS')]).

ctr_example(
    elements_sparse,
    elements_sparse(
        [[index-8, value-9],
         [index-3, value-5],
         [index-2, value-5]],
        [[index-1, value-6],
         [index-2, value-5],
         [index-4, value-2],

```


[index-8,value-9]],
5)).

B.89 eq_set

```
ctr_predefined(eq_set).  
  
ctr_date(eq_set, ['20030820']).  
  
ctr_origin(  
    eq_set,  
    'Used for defining %c.',  
    [alldifferent_between_sets]).  
  
ctr_arguments(eq_set, ['SET1'-svar, 'SET2'-svar]).  
  
ctr_example(eq_set, eq_set({3,5}, {3,5})).
```

B.90 exactly

```

ctr_automaton(exactly, exactly) .

ctr_date(exactly, ['20040807']) .

ctr_origin(exactly, 'Derived from %c and %c.', [atleast, atmost]) .

ctr_arguments(
    exactly,
    ['N'-int, 'VARIABLES'-collection(var-dvar), 'VALUE'-int]) .

ctr_restrictions(
    exactly,
    ['N'>=0, 'N'<=size('VARIABLES'), required('VARIABLES', var)]) .

ctr_graph(
    exactly,
    ['VARIABLES'],
    1,
    ['SELF'>>collection(variables)],
    [variables^var='VALUE'],
    ['NARC'='N']) .

ctr_example(
    exactly,
    exactly(2, [[var-4], [var-2], [var-4], [var-5]], 4)) .

exactly(A, B, C) :-
    exactly_signature(B, D, C),
    automaton(
        D,
        E,
        D,
        0..1,
        [source(s), sink(t)],
        [arc(s, 0, s), arc(s, 1, s, [F+1]), arc(s, $, t)],
        [F],
        [0],
        [A]) .

exactly_signature([], [], A) .

exactly_signature([[var-A] | B], [C | D], E) :-
    A#=E#<=>C,
    exactly_signature(B, D, E) .

```


B.91 global_cardinality

```
ctr_date(global_cardinality, ['20030820', '20040530']).
```

```
ctr_origin(global_cardinality, 'CHARME', []).
```

```
ctr_synonyms(
  global_cardinality,
  [distribute, distribution, gcc, card_var_gcc, egcc]).
```

```
ctr_arguments(
  global_cardinality,
  ['VARIABLES'-collection(var-dvar),
   'VALUES'-collection(val-int, noccurrence-dvar)]).
```

```
ctr_restrictions(
  global_cardinality,
  [required('VARIABLES', var),
   required('VALUES', [val, noccurrence]),
   distinct('VALUES', val),
   'VALUES'^noccurrence>=0,
   'VALUES'^noccurrence=<size('VARIABLES')]).
```

```
ctr_graph(
  global_cardinality,
  ['VARIABLES'],
  1,
  foreach('VALUES', ['SELF'>>collection(variables)]),
  [variables^var='VALUES'^val],
  ['NVERTEX'='VALUES'^noccurrence]).
```

```
ctr_example(
  global_cardinality,
  global_cardinality(
    [[var-3], [var-3], [var-8], [var-6]],
    [[val-3, noccurrence-2],
     [val-5, noccurrence-0],
     [val-6, noccurrence-1]])).
```

B.92 global_cardinality_low_up

```

ctr_date(global_cardinality_low_up, ['20031008', '20040530']).

ctr_origin(
    global_cardinality_low_up,
    'Used for defining %c.',
    [sliding_distribution]).

ctr_arguments(
    global_cardinality_low_up,
    ['VARIABLES'-collection(var-dvar),
     'VALUES'-collection(val-int, omin-int, omax-int)]).

ctr_restrictions(
    global_cardinality_low_up,
    [required('VARIABLES', var),
     size('VALUES')>0,
     required('VALUES', [val, omin, omax]),
     distinct('VALUES', val),
     'VALUES'^omin>=0,
     'VALUES'^omax<=size('VARIABLES'),
     'VALUES'^omin<='VALUES'^omax]).

ctr_graph(
    global_cardinality_low_up,
    ['VARIABLES'],
    1,
    foreach('VALUES', ['SELF']>>collection(variables)),
    [variables^var='VALUES'^val],
    ['NVERTEX'>='VALUES'^omin, 'NVERTEX'<='VALUES'^omax]).

ctr_example(
    global_cardinality_low_up,
    global_cardinality_low_up(
        [[var-3], [var-3], [var-8], [var-6]],
        [[val-3, omin-2, omax-3],
         [val-5, omin-0, omax-1],
         [val-6, omin-1, omax-2]])).

```

B.93 global_cardinality_with_costs

```
ctr_date(global_cardinality_with_costs, ['20030820', '20040530']).
```

```
ctr_origin(global_cardinality_with_costs, '\\cite{Regin99a}', []).
```

```
ctr_synonyms(global_cardinality_with_costs, [gccc, cost_gcc]).
```

```
ctr_arguments(
  global_cardinality_with_costs,
  ['VARIABLES'-collection(var-dvar),
   'VALUES'-collection(val-int, noccurrence-dvar),
   'MATRIX'-collection(i-int, j-int, c-int),
   'COST'-dvar]).
```

```
ctr_restrictions(
  global_cardinality_with_costs,
  [required('VARIABLES', var),
   required('VALUES', [val, noccurrence]),
   distinct('VALUES', val),
   'VALUES' ^ noccurrence >= 0,
   'VALUES' ^ noccurrence <= size('VARIABLES'),
   required('MATRIX', [i, j, c]),
   increasing_seq('MATRIX', [i, j]),
   'MATRIX' ^ i >= 1,
   'MATRIX' ^ i <= size('VARIABLES'),
   'MATRIX' ^ j >= 1,
   'MATRIX' ^ j <= size('VALUES'),
   size('MATRIX') = size('VARIABLES') * size('VALUES')]).
```

```
ctr_graph(
  global_cardinality_with_costs,
  ['VARIABLES'],
  1,
  foreach('VALUES', ['SELF' >> collection(variables)]),
  [variables ^ var = 'VALUES' ^ val],
  ['NVERTEX' = 'VALUES' ^ noccurrence]).
```

```
ctr_graph(
  global_cardinality_with_costs,
  ['VARIABLES', 'VALUES'],
  2,
  ['PRODUCT' >> collection(variables, values)],
  [variables ^ var = values ^ val],
  [= ('SUM_WEIGHT_ARC' (
    ^ (@('MATRIX',
```

```

        + ((variables^key-1)*size('VALUES'),
          values^key)),
      c)),
    'COST')]).

```

```

ctr_example(
  global_cardinality_with_costs,
  global_cardinality_with_costs(
    [[var-3],[var-3],[var-3],[var-6]],
    [[val-3,noccurrence-3],
     [val-5,noccurrence-0],
     [val-6,noccurrence-1]],
    [[i-1,j-1,c-4],
     [i-1,j-2,c-1],
     [i-1,j-3,c-7],
     [i-2,j-1,c-1],
     [i-2,j-2,c-0],
     [i-2,j-3,c-8],
     [i-3,j-1,c-3],
     [i-3,j-2,c-2],
     [i-3,j-3,c-1],
     [i-4,j-1,c-0],
     [i-4,j-2,c-0],
     [i-4,j-3,c-6]],
    14)).

```


B.94 global_contiguity

```

ctr_automaton(global_contiguity,global_contiguity).
ctr_date(global_contiguity,['20030820','20040530']).
ctr_origin(global_contiguity,'\\cite{Maher02}',[]).
ctr_arguments(
    global_contiguity,
    ['VARIABLES'-collection(var-dvar)]).
ctr_restrictions(
    global_contiguity,
    [required('VARIABLES',var),
     'VARIABLES'^var>=0,
     'VARIABLES'^var=<1]).
ctr_graph(
    global_contiguity,
    ['VARIABLES'],
    2,
    ['PATH'>>collection(variables1,variables2),
     'LOOP'>>collection(variables1,variables2)],
    [variables1^var=variables2^var,variables1^var=1],
    ['NCC'=<1]).
ctr_example(
    global_contiguity,
    global_contiguity([[var-0],[var-1],[var-1],[var-0]])).
global_contiguity(A) :-
    col_to_list(A,B),
    automaton(
        B,
        C,
        B,
        0..1,
        [source(s),node(n),node(z),sink(t)],
        [arc(s,0,s),
         arc(s,1,n),
         arc(s,$,t),
         arc(n,0,z),
         arc(n,1,n),
         arc(n,$,t),
         arc(z,0,z),

```

```
arc(z, $, t)],  
[],  
[],  
[]).
```

B.95 golomb

```
ctr_date(golomb, ['20000128', '20030820', '20040530']).
ctr_origin(golomb, 'Inspired by \\cite{Golomb72}.'. , []).
ctr_arguments(golomb, ['VARIABLES'-collection(var-dvar)]).
ctr_restrictions(
    golomb,
    [required('VARIABLES', var), 'VARIABLES'^var>=0]).
ctr_derived_collections(
    golomb,
    [col('PAIRS'-collection(x-dvar, y-dvar),
        [> -item(x-'VARIABLES'^var, y-'VARIABLES'^var)])]).
ctr_graph(
    golomb,
    ['PAIRS'],
    2,
    ['CLIQUE'>>collection(pairs1, pairs2)],
    [pairs1^y-pairs1^x=pairs2^y-pairs2^x],
    ['MAX_NSCC'=<1]).
ctr_example(golomb, golomb([[var-0], [var-1], [var-4], [var-6]])).
```

B.96 graph_crossing

```

ctr_date(graph_crossing, ['20000128', '20030820', '20040530']).

ctr_origin(graph_crossing, 'N.~Beldiceanu', []).

ctr_arguments(
    graph_crossing,
    ['NCROSS'-dvar, 'NODES'-collection(succ-dvar, x-int, y-int)]).

ctr_restrictions(
    graph_crossing,
    ['NCROSS'>=0,
     required('NODES', [succ, x, y]),
     'NODES'^succ>=1,
     'NODES'^succ<=size('NODES')]).

ctr_graph(
    graph_crossing,
    ['NODES'],
    2,
    ['CLIQUE' (<)>>collection(n1, n2)],
    [>=(max(n1^x, 'NODES'@(n1^succ)^x),
       min(n2^x, 'NODES'@(n2^succ)^x)),
     >=(max(n2^x, 'NODES'@(n2^succ)^x),
       min(n1^x, 'NODES'@(n1^succ)^x)),
     >=(max(n1^y, 'NODES'@(n1^succ)^y),
       min(n2^y, 'NODES'@(n2^succ)^y)),
     >=(max(n2^y, 'NODES'@(n2^succ)^y),
       min(n1^y, 'NODES'@(n1^succ)^y)),
     =\=(-(* (n2^x-'NODES'@(n1^succ)^x,
             'NODES'@(n1^succ)^y-n1^y),
          * ('NODES'@(n1^succ)^x-n1^x,
            n2^y-'NODES'@(n1^succ)^y)),
        0),
     =\=(-(* ('NODES'@(n2^succ)^x-'NODES'@(n1^succ)^x,
             n2^y-n1^y),
          * (n2^x-n1^x,
            'NODES'@(n2^succ)^y-'NODES'@(n1^succ)^y)),
        0),
     =\=(sign(
         -(* (n2^x-'NODES'@(n1^succ)^x,
             'NODES'@(n1^succ)^y-n1^y),
          * ('NODES'@(n1^succ)^x-n1^x,
            n2^y-'NODES'@(n1^succ)^y)),
         sign(

```

```

- (* ('NODES' @ (n2^succ) ^x-'NODES' @ (n1^succ) ^x,
      n2^y-n1^y),
   * (n2^x-n1^x,
      'NODES' @ (n2^succ) ^y-'NODES' @ (n1^succ) ^y)))],
['NARC'='NCROSS'] ).

ctr_example(
  graph_crossing,
  graph_crossing(
    2,
    [[succ-1, x-4, y-7],
     [succ-1, x-2, y-5],
     [succ-1, x-7, y-6],
     [succ-2, x-1, y-2],
     [succ-3, x-2, y-2],
     [succ-2, x-5, y-3],
     [succ-3, x-8, y-2],
     [succ-9, x-6, y-2],
     [succ-10, x-10, y-6],
     [succ-8, x-10, y-1]])).

```

B.97 group

```
ctr_automaton(group, group) .
```

```
ctr_date(group, ['20000128', '20030820', '20040530']) .
```

```
ctr_origin(group, 'CHIP', []).
```

```
ctr_arguments(
  group,
  ['NGROUP'-dvar,
   'MIN_SIZE'-dvar,
   'MAX_SIZE'-dvar,
   'MIN_DIST'-dvar,
   'MAX_DIST'-dvar,
   'NVAL'-dvar,
   'VARIABLES'-collection(var-dvar),
   'VALUES'-collection(val-int)].
```

```
ctr_restrictions(
  group,
  ['NGROUP'>=0,
   'MIN_SIZE'>=0,
   'MAX_SIZE'>='MIN_SIZE',
   'MIN_DIST'>=0,
   'MAX_DIST'>='MIN_DIST',
   'NVAL'>=0,
   required('VARIABLES', var),
   required('VALUES', val),
   distinct('VALUES', val)].
```

```
ctr_graph(
  group,
  ['VARIABLES'],
  2,
  ['PATH'>>collection(variables1, variables2),
   'LOOP'>>collection(variables1, variables2)],
  [in(variables1^var, 'VALUES'), in(variables2^var, 'VALUES')],
  ['NCC'='NGROUP',
   'MIN_NCC'='MIN_SIZE',
   'MAX_NCC'='MAX_SIZE',
   'NVERTEX'='NVAL']).
```

```
ctr_graph(
  group,
  ['VARIABLES'],
```

```

2,
['PATH'>>collection(variables1,variables2),
 'LOOP'>>collection(variables1,variables2)],
[not_in(variables1^var,'VALUES'),
 not_in(variables2^var,'VALUES')],
['MIN_NCC'='MIN_DIST','MAX_NCC'='MAX_DIST'] ).

ctr_example(
  group,
  group(
    2,
    1,
    2,
    2,
    4,
    3,
    [[var-2],
     [var-8],
     [var-1],
     [var-7],
     [var-4],
     [var-5],
     [var-1],
     [var-1],
     [var-1]],
    [[val-0],[val-2],[val-4],[val-6],[val-8]]) ).

group(A,B,C,D,E,F,G,H) :-
  group_ngroup(A,G,H),
  group_min_size(B,G,H),
  group_max_size(C,G,H),
  group_min_dist(D,G,H),
  group_max_dist(E,G,H),
  group_nval(F,G,H).

group_ngroup(A,B,C) :-
  col_to_list(C,D),
  list_to_fdset(D,E),
  group_signature_in(B,F,E),
  automaton(
    F,
    G,
    F,
    0..1,
    [source(s),node(i),sink(t)],
    [arc(s,0,s),

```

```

    arc(s,1,i,[H+1]),
    arc(s,$,t),
    arc(i,1,i),
    arc(i,0,s),
    arc(i,$,t)],
[H],
[0],
[A]).

```

```

group_min_size(A,B,C) :-
    length(B,D),
    col_to_list(C,E),
    list_to_fdset(E,F),
    group_signature_in(B,G,F),
    automaton(
        G,
        H,
        G,
        0..1,
        [source(s),node(j),node(k),sink(t)],
        [arc(s,0,s),
        arc(s,1,j,[D,I]),
        arc(s,$,t),
        arc(j,1,j,[J,I+1]),
        arc(j,0,k,[min(J,I),I]),
        arc(j,$,t,[min(J,I),I]),
        arc(k,0,k),
        arc(k,1,j,[J,1]),
        arc(k,$,t)],
        [J,I],
        [0,1],
        [A,K]).

```

```

group_max_size(A,B,C) :-
    col_to_list(C,D),
    list_to_fdset(D,E),
    group_signature_in(B,F,E),
    automaton(
        F,
        G,
        F,
        0..1,
        [source(s),sink(t)],
        [arc(s,1,s,[H,I+1]),
        arc(s,0,s,[max(H,I),0]),
        arc(s,$,t,[max(H,I),I])],

```



```

[H, I],
[0, 0],
[A, J]).

```

```

group_min_dist(A,B,C) :-
    length(B,D),
    col_to_list(C,E),
    list_to_fdset(E,F),
    group_signature_not_in(B,G,F),
    automaton(
        G,
        H,
        G,
        0..1,
        [source(s), node(j), node(k), sink(t)],
        [arc(s,0,s),
         arc(s,1,j,[D,I]),
         arc(s,$,t),
         arc(j,1,j,[J,I+1]),
         arc(j,0,k,[min(J,I),I]),
         arc(j,$,t,[min(J,I),I]),
         arc(k,0,k),
         arc(k,1,j,[J,1]),
         arc(k,$,t)],
        [J,I],
        [0,1],
        [A,K]).

```

```

group_max_dist(A,B,C) :-
    col_to_list(C,D),
    list_to_fdset(D,E),
    group_signature_not_in(B,F,E),
    automaton(
        F,
        G,
        F,
        0..1,
        [source(s), sink(t)],
        [arc(s,1,s,[H,I+1]),
         arc(s,0,s,[max(H,I),0]),
         arc(s,$,t,[max(H,I),I])],
        [H,I],
        [0,0],
        [A,J]).

```

```

group_nval(A,B,C) :-

```

```

col_to_list(C,D),
list_to_fdset(D,E),
group_signature_in(B,F,E),
automaton(
    F,
    G,
    F,
    0..1,
    [source(s),sink(t)],
    [arc(s,0,s),arc(s,1,s,[H+1]),arc(s,$,t)],
    [H],
    [0],
    [A]).

```

```
group_signature_in([],[],A).
```

```

group_signature_in([[var-A]|B],[C|D],E) :-
    in_set(A,E) #<=> C,
    group_signature_in(B,D,E).

```

```
group_signature_not_in([],[],A).
```

```

group_signature_not_in([[var-A]|B],[C|D],E) :-
    in_set(A,E) #<=> #\C,
    group_signature_not_in(B,D,E).

```

B.98 group_skip_isolated_item

```

ctr_automaton(
    group_skip_isolated_item,
    group_skip_isolated_item).

ctr_date(
    group_skip_isolated_item,
    ['20000128', '20030820', '20040530']).

ctr_origin(group_skip_isolated_item, 'Derived from %c.', [group]).

ctr_arguments(
    group_skip_isolated_item,
    ['NGROUP'-dvar,
     'MIN_SIZE'-dvar,
     'MAX_SIZE'-dvar,
     'NVAL'-dvar,
     'VARIABLES'-collection(var-dvar),
     'VALUES'-collection(val-int)]).

ctr_restrictions(
    group_skip_isolated_item,
    ['NGROUP'>=0,
     'MIN_SIZE'>=0,
     'MAX_SIZE'>='MIN_SIZE',
     'NVAL'>=0,
     required('VARIABLES', var),
     required('VALUES', val),
     distinct('VALUES', val)]).

ctr_graph(
    group_skip_isolated_item,
    ['VARIABLES'],
    2,
    ['CHAIN'>>collection(variables1, variables2)],
    [in(variables1^var, 'VALUES'), in(variables2^var, 'VALUES')],
    ['NSCC'='NGROUP',
     'MIN_NSCC'='MIN_SIZE',
     'MAX_NSCC'='MAX_SIZE',
     'NVERTEX'='NVAL'] ).

ctr_example(
    group_skip_isolated_item,
    group_skip_isolated_item(
        1,

```

```

2,
2,
3,
[[var-2],
 [var-8],
 [var-1],
 [var-7],
 [var-4],
 [var-5],
 [var-1],
 [var-1],
 [var-1]],
[[val-0],[val-2],[val-4],[val-6],[val-8]])).

```

```

group_skip_isolated_item(A,B,C,D,E,F) :-
    group_skip_isolated_item_ngroup(A,E,F),
    group_skip_isolated_item_min_size(B,E,F),
    group_skip_isolated_item_max_size(C,E,F),
    group_skip_isolated_item_nval(D,E,F).

```

```

group_skip_isolated_item_ngroup(A,B,C) :-
    col_to_list(C,D),
    list_to_fdset(D,E),
    group_skip_isolated_item_signature(B,F,E),
    automaton(
        F,
        G,
        F,
        0..1,
        [source(s),node(i),node(j),sink(t)],
        [arc(s,0,s),
         arc(s,1,i),
         arc(s,$,t),
         arc(i,0,s),
         arc(i,1,j,[H+1]),
         arc(i,$,t),
         arc(j,1,j),
         arc(j,0,s),
         arc(j,$,t)],
        [H],
        [0],
        [A]).

```

```

group_skip_isolated_item_min_size(A,B,C) :-
    length(B,D),
    col_to_list(C,E),

```

```

list_to_fdset(E,F),
group_skip_isolated_item_signature(B,G,F),
automaton(
  G,
  H,
  G,
  0..1,
  [source(s),
   node(j),
   node(k),
   node(l),
   node(m),
   sink(t)],
  [arc(s,0,s),
   arc(s,1,j),
   arc(s,$,t),
   arc(j,0,s),
   arc(j,1,k,[D,I]),
   arc(j,$,t),
   arc(k,1,k,[J,I+1]),
   arc(k,0,l,[min(J,I),I]),
   arc(k,$,t,[min(J,I),I]),
   arc(l,0,l),
   arc(l,1,m),
   arc(l,$,t),
   arc(m,0,l),
   arc(m,1,k,[J,2]),
   arc(m,$,t)],
  [J,I],
  [0,2],
  [A,K]).

```

```

group_skip_isolated_item_max_size(A,B,C) :-
  col_to_list(C,D),
  list_to_fdset(D,E),
  group_skip_isolated_item_signature(B,F,E),
  automaton(
    F,
    G,
    F,
    0..1,
    [source(s),node(i),sink(t)],
    [arc(s,0,s),
     arc(s,1,i,[H,1]),
     arc(s,$,t),
     arc(i,0,s,[max(H,I),I]),

```

```

    arc(i,1,i,[H,I+1]),
    arc(i,$,t,[max(H,I),I]),
    [H,I],
    [0,0],
    [A,J]).

```

```

group_skip_isolated_item_nval(A,B,C) :-
    col_to_list(C,D),
    list_to_fdset(D,E),
    group_skip_isolated_item_signature(B,F,E),
    automaton(
        F,
        G,
        F,
        0..1,
        [source(s),sink(t)],
        [arc(s,0,s),arc(s,1,s,[H+1]),arc(s,$,t)],
        [H],
        [0],
        [A]).

```

```

group_skip_isolated_item_signature([],[],A).

```

```

group_skip_isolated_item_signature([[var-A]|B],[C|D],E) :-
    in_set(A,E)#<=>C,
    group_skip_isolated_item_signature(B,D,E).

```

B.99 highest_peak

```

ctr_automaton(highest_peak,highest_peak) .

ctr_date(highest_peak,['20040530']).

ctr_origin(highest_peak,'Derived from %c.',[peak]).

ctr_arguments(
    highest_peak,
    ['HEIGHT'-dvar,'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    highest_peak,
    ['HEIGHT'>=0,'VARIABLES'^var>=0,required('VARIABLES',var)]).

ctr_example(
    highest_peak,
    highest_peak(
        8,
        [[var-1],
         [var-1],
         [var-4],
         [var-8],
         [var-6],
         [var-2],
         [var-7],
         [var-1]])).

highest_peak(A,B) :-
    highest_peak_signature(B,C,D),
    automaton(
        D,
        E-F,
        C,
        0..2,
        [source(s),node(u),sink(t)],
        [arc(s,0,s),
         arc(s,1,s),
         arc(s,2,u),
         arc(s,$,t),
         arc(u,0,s,[max(G,E)]),
         arc(u,1,u),
         arc(u,2,u),
         arc(u,$,t)],
        [G],

```

```
[0],  
[A]).
```

```
highest_peak_signature([], [], []).
```

```
highest_peak_signature([A], [], []).
```

```
highest_peak_signature([[var-A], [var-B] | C], [D | E], [A-B | F]) :-  
    in(D, 0..2),  
    A#>B#<=>D#=0,  
    A#=B#<=>D#=1,  
    A#<B#<=>D#=2,  
    highest_peak_signature([var-B] | C, E, F).
```


B.100 in

```

ctr_automaton(in, in_).

ctr_date(in, ['20030820', '20040530']).

ctr_origin(in, 'Domain definition.', []).

ctr_arguments(in, ['VAR'-dvar, 'VALUES'-collection(val-int)]).

ctr_restrictions(
    in,
    [required('VALUES', val), distinct('VALUES', val)]).

ctr_derived_collections(
    in,
    [col('VARIABLES'-collection(var-dvar), [item(var-'VAR')])]).

ctr_graph(
    in,
    ['VARIABLES', 'VALUES'],
    2,
    ['PRODUCT' >> collection(variables, values)],
    [variables^var=values^val],
    ['NARC'=1]).

ctr_example(in, in(3, [[val-1], [val-3]])).

in_(A, B) :-
    in_signature(B, C, A),
    automaton(
        C,
        D,
        C,
        0..1,
        [source(s), sink(t)],
        [arc(s, 0, s), arc(s, 1, t)],
        [],
        [],
        []).

in_signature([], [], A).

in_signature([[val-A]|B], [C|D], E) :-
    E#=A#<=>C,
    in_signature(B, D, E).

```


B.101 in_relation

```

ctr_date(in_relation, ['20030820', '20040530']).

ctr_origin(
    in_relation,
    'Constraint explicitly defined by tuples of values.',
    []).

ctr_synonyms(in_relation, [extension]).

ctr_types(
    in_relation,
    ['TUPLE_OF_VARS'-collection(var-dvar),
     'TUPLE_OF_VALS'-collection(val-int)]).

ctr_arguments(
    in_relation,
    ['VARIABLES'-'TUPLE_OF_VARS',
     'TUPLES_OF_VALS'-collection(tuple-'TUPLE_OF_VALS')]).

ctr_restrictions(
    in_relation,
    [required('TUPLE_OF_VARS', var),
     required('TUPLE_OF_VALS', val),
     required('TUPLES_OF_VALS', tuple),
     min_size('TUPLES_OF_VALS', tuple)=size('VARIABLES'),
     max_size('TUPLES_OF_VALS', tuple)=size('VARIABLES')]).

ctr_derived_collections(
    in_relation,
    [col('TUPLES_OF_VARS'-collection(vec-'TUPLE_OF_VARS'),
        [item(vec-'VARIABLES')])]).

ctr_graph(
    in_relation,
    ['TUPLES_OF_VARS', 'TUPLES_OF_VALS'],
    2,
    ['PRODUCT'>>collection(tuples_of_vars, tuples_of_vals)],
    [vec_eq_tuple(tuples_of_vars^vec, tuples_of_vals^tuple)],
    ['NARC'>=1]).

ctr_example(
    in_relation,
    in_relation(
        [[var-5], [var-3], [var-3]],

```

```
[[tuple-[val-5],[val-2],[val-3]]],  
 [tuple-[val-5],[val-2],[val-6]]],  
 [tuple-[val-5],[val-3],[val-3]]]])).
```

B.102 in_same_partition

```

ctr_automaton(in_same_partition,in_same_partition).

ctr_date(in_same_partition,['20030820','20040530']).

ctr_origin(
    in_same_partition,
    'Used for defining several entries of this catalog.',
    []).

ctr_types(in_same_partition,['VALUES'-collection(val-int)]).

ctr_arguments(
    in_same_partition,
    ['VAR1'-dvar,
     'VAR2'-dvar,
     'PARTITIONS'-collection(p-'VALUES')]).

ctr_restrictions(
    in_same_partition,
    [required('VALUES',val),
     distinct('VALUES',val),
     required('PARTITIONS',p),
     size('PARTITIONS')>=2]).

ctr_derived_collections(
    in_same_partition,
    [col('VARIABLES'-collection(var-dvar),
        [item(var-'VAR1'),item(var-'VAR2')])]).

ctr_graph(
    in_same_partition,
    ['VARIABLES','PARTITIONS'],
    2,
    ['PRODUCT'>>collection(variables,partitions)],
    [in(variables^var,partitions^p)],
    ['NSOURCE'=2,'NSINK'=1]).

ctr_example(
    in_same_partition,
    in_same_partition(
        6,
        2,
        [p-[[val-1],[val-3]]],
        [p-[[val-4]]],

```

```

    [p-[[val-2],[val-6]]]))).

in_same_partition(A,B,C) :-
    in_same_partition_signature(C,D,A,B),
    automaton(
        D,
        E,
        D,
        0..1,
        [source(s),sink(t)],
        [arc(s,0,s),arc(s,1,t)],
        [],
        [],
        []).

in_same_partition_signature([],[],A,B).

in_same_partition_signature([[p-A]|B],[C|D],E,F) :-
    col_to_list(A,G),
    list_to_fdset(G,H),
    in_set(E,H)#/\in_set(F,H)#<=>C,
    in_same_partition_signature(B,D,E,F).

```

B.103 in_set

```
ctr_predefined(in_set).
```

```
ctr_date(in_set, ['20030820']).
```

```
ctr_origin(  
    in_set,  
    'Used for defining constraints with set variables.',  
    []).
```

```
ctr_arguments(in_set, ['VAL'-dvar, 'SET'-svar]).
```

```
ctr_example(in_set, in_set(3, {1, 3})).
```

B.104 increasing

```

ctr_automaton(increasing, increasing) .

ctr_date(increasing, ['20040814']) .

ctr_origin(increasing, 'KOALOG', []).

ctr_arguments(increasing, ['VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    increasing,
    [size('VARIABLES')>0, required('VARIABLES', var)]).

ctr_graph(
    increasing,
    ['VARIABLES'],
    2,
    ['PATH'>>collection(variables1, variables2)],
    [variables1^var=<variables2^var],
    ['NARC'=size('VARIABLES')-1]).

ctr_example(
    increasing,
    increasing([[var-1], [var-1], [var-4], [var-8]])).

increasing(A) :-
    increasing_signature(A, B),
    automaton(
        B,
        C,
        B,
        0..1,
        [source(s), sink(t)],
        [arc(s, 0, s), arc(s, $, t)],
        [],
        [],
        []).

increasing_signature([A], []).

increasing_signature([[var-A], [var-B] | C], [D | E]) :-
    in(D, 0..1),
    A#>B#<=>D,
    increasing_signature([[var-B] | C], E).

```


B.105 indexed_sum

```

ctr_date(indexed_sum, ['20040814']).

ctr_origin(indexed_sum, 'N.~Beldiceanu', []).

ctr_arguments(
    indexed_sum,
    ['ITEMS'-collection(index-dvar, weight-dvar),
     'TABLE'-collection(index-int, sum-dvar)]).

ctr_restrictions(
    indexed_sum,
    [size('ITEMS')>0,
     size('TABLE')>0,
     required('ITEMS', [index, weight]),
     'ITEMS'^index>=0,
     'ITEMS'^index<size('TABLE'),
     required('TABLE', [index, sum]),
     'TABLE'^index>=0,
     'TABLE'^index<size('TABLE'),
     increasing_seq('TABLE', index)]).

ctr_graph(
    indexed_sum,
    ['ITEMS', 'TABLE'],
    2,
    foreach('TABLE', ['PRODUCT'>>collection(items, table)]),
    [items^index=table^index],
    [],
    [>>('SUCC',
        [source,
         -(variables,
            col('VARIABLES'-collection(var-dvar),
              [item(var-'ITEMS'^weight)]))]],
        [sum_ctr(variables, =, 'TABLE'^sum)]).

ctr_example(
    indexed_sum,
    indexed_sum(
        [[index-2, weight- -4],
         [index-0, weight-6],
         [index-2, weight-1]],
        [[index-0, sum-6], [index-1, sum-0], [index-2, sum- -3]]).

```

B.106 inflexion

```

ctr_automaton(inflexion, inflexion) .

ctr_date(inflexion, ['20000128', '20030820', '20040530']) .

ctr_origin(inflexion, 'N.~Beldiceanu', []).

ctr_arguments(
    inflexion,
    ['N'-dvar, 'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    inflexion,
    ['N'>=1, 'N'=<size('VARIABLES'), required('VARIABLES', var)]).

ctr_example(
    inflexion,
    inflexion(
        3,
        [[var-1],
         [var-1],
         [var-4],
         [var-8],
         [var-8],
         [var-2],
         [var-7],
         [var-1]])).

inflexion(A,B) :-
    inflexion_signature(B,C),
    automaton(
        C,
        D,
        C,
        0..2,
        [source(s), node(i), node(j), sink(t)],
        [arc(s,1,s),
         arc(s,2,i),
         arc(s,0,j),
         arc(s,$,t),
         arc(i,1,i),
         arc(i,2,i),
         arc(i,0,j,[E+1]),
         arc(i,$,t),
         arc(j,1,j),

```

```
arc(j,0,j),
arc(j,2,i,[E+1]),
arc(j,$,t)],
[E],
[0],
[A]).
```

```
inflexion_signature([],[]).
```

```
inflexion_signature([A],[]).
```

```
inflexion_signature([[var-A],[var-B]|C],[D|E]) :-
in(D,0..2),
A#>B#<=>D#=0,
A#=B#<=>D#=1,
A#<B#<=>D#=2,
inflexion_signature([[var-B]|C],E).
```

B.107 int_value_precede

```

ctr_automaton(int_value_precede,int_value_precede).

ctr_date(int_value_precede,['20041003']).

ctr_origin(int_value_precede,'\cite{YatChiuLawJimmyLee04}',[]).

ctr_arguments(
    int_value_precede,
    ['S'-int,'T'-int,'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    int_value_precede,
    ['S'='\='T',required('VARIABLES',var)]).

ctr_example(
    int_value_precede,
    int_value_precede(
        0,
        1,
        [[var-4],[var-0],[var-6],[var-1],[var-0]]).

int_value_precede(A,B,C) :-
    int_value_precede_signature(C,D,A,B),
    automaton(
        D,
        E,
        D,
        1..3,
        [source(s),sink(t)],
        [arc(s,3,s),arc(s,1,t),arc(s,$,t)],
        [],
        [],
        []).

int_value_precede_signature([],[],A,B).

int_value_precede_signature([[var-A]|B],[C|D],E,F) :-
    in(C,1..3),
    A#=E#<=>C#=1,
    A#=F#<=>C#=2,
    A#\=E#\A#\=F#<=>C#=3,
    int_value_precede_signature(B,D,E,F).

```

B.108 int_value_precede_chain

```
ctr_automaton(int_value_precede_chain,int_value_precede_chain).  
  
ctr_date(int_value_precede_chain,['20041003']).  
  
ctr_origin(  
    int_value_precede_chain,  
    '\\cite{YatChiuLawJimmyLee04}',  
    []).  
  
ctr_arguments(  
    int_value_precede_chain,  
    ['VALUES'-collection(val-int),  
     'VARIABLES'-collection(var-dvar)]).  
  
ctr_restrictions(  
    int_value_precede_chain,  
    [required('VALUES',val),  
     distinct('VALUES',val),  
     required('VARIABLES',var)]).  
  
ctr_example(  
    int_value_precede_chain,  
    int_value_precede_chain(  
        [[val-4],[val-0],[val-1]],  
        [[var-4],[var-0],[var-6],[var-1],[var-0]])).  
  
int_value_precede_chain(A,B).
```

B.109 interval_and_count

```
ctr_date(interval_and_count, ['20000128', '20030820', '20040530']).
```

```
ctr_origin(interval_and_count, '\\cite{Cousin93}', []).
```

```
ctr_arguments(
  interval_and_count,
  ['ATMOST'-int,
   'COLOURS'-collection(val-int),
   'TASKS'-collection(origin-dvar, colour-dvar),
   'SIZE_INTERVAL'-int]).
```

```
ctr_restrictions(
  interval_and_count,
  ['ATMOST'>=0,
   required('COLOURS', val),
   distinct('COLOURS', val),
   required('TASKS', [origin, colour]),
   'SIZE_INTERVAL'>0]).
```

```
ctr_graph(
  interval_and_count,
  ['TASKS', 'TASKS'],
  2,
  ['PRODUCT'>>collection(tasks1, tasks2)],
  [(tasks1^origin/'SIZE_INTERVAL',
    tasks2^origin/'SIZE_INTERVAL')],
  [],
  [>>('SUCC',
    [source,
     -(variables,
      col('VARIABLES'-collection(var-dvar),
        [item(var-'TASKS'^colour)]))]),
  [among_low_up(0, 'ATMOST', variables, 'COLOURS')]).
```

```
ctr_example(
  interval_and_count,
  interval_and_count(
    2,
    [[val-4]],
    [[origin-1, colour-4],
     [origin-0, colour-9],
     [origin-10, colour-4],
     [origin-4, colour-4]],
    5)).
```


B.110 interval_and_sum

```

ctr_date(interval_and_sum, ['20000128', '20030820']).

ctr_origin(interval_and_sum, 'Derived from %c.', [cumulative]).

ctr_arguments(
    interval_and_sum,
    ['SIZE_INTERVAL'-int,
     'TASKS'-collection(origin-dvar, height-dvar),
     'LIMIT'-int]).

ctr_restrictions(
    interval_and_sum,
    ['SIZE_INTERVAL'>0,
     required('TASKS', [origin, height]),
     'TASKS'^height>=0,
     'LIMIT'>=0]).

ctr_graph(
    interval_and_sum,
    ['TASKS', 'TASKS'],
    2,
    ['PRODUCT'>>collection(tasks1, tasks2)],
    [(tasks1^origin/'SIZE_INTERVAL',
      tasks2^origin/'SIZE_INTERVAL')],
    [],
    [>>('SUCC',
        [source,
         -(variables,
            col('VARIABLES'-collection(var-dvar),
              [item(var-'TASKS'^height)]))]),
     [sum_ctr(variables, =<, 'LIMIT')]).

ctr_example(
    interval_and_sum,
    interval_and_sum(
        5,
        [[origin-1, height-2],
         [origin-10, height-2],
         [origin-10, height-3],
         [origin-4, height-1]],
        5)).

```


B.111 inverse

```

ctr_date(inverse, ['20000128', '20030820', '20040530']).

ctr_origin(inverse, 'CHIP', []).

ctr_synonyms(inverse, [assignment]).

ctr_arguments(
    inverse,
    ['NODES'-collection(index-int, succ-dvar, pred-dvar)]).

ctr_restrictions(
    inverse,
    [required('NODES', [index, succ, pred]),
     'NODES'^index>=1,
     'NODES'^index<=size('NODES'),
     distinct('NODES', index),
     'NODES'^succ>=1,
     'NODES'^succ<=size('NODES'),
     'NODES'^pred>=1,
     'NODES'^pred<=size('NODES')]).

ctr_graph(
    inverse,
    ['NODES'],
    2,
    ['CLIQUE'>>collection(nodes1, nodes2)],
    [nodes1^succ=nodes2^index, nodes2^pred=nodes1^index],
    ['NARC'=size('NODES')]).

ctr_example(
    inverse,
    inverse(
        [[index-1, succ-2, pred-2],
         [index-2, succ-1, pred-1],
         [index-3, succ-5, pred-4],
         [index-4, succ-3, pred-5],
         [index-5, succ-4, pred-3]])).

```

B.112 inverse_set

```

ctr_date(inverse_set, ['20041211']).

ctr_origin(inverse_set, 'Derived from %c.', [inverse]).

ctr_arguments(
    inverse_set,
    ['X'-collection(index-int, set-svar),
     'Y'-collection(index-int, set-svar)]).

ctr_restrictions(
    inverse_set,
    [required('X', [index, set]),
     required('Y', [index, set]),
     increasing_seq('X', index),
     increasing_seq('Y', index),
     'X'^index>=1,
     'X'^index<=size('Y'),
     'Y'^index>=1,
     'Y'^index<=size('X'),
     'X'^set>=1,
     'X'^set<=size('Y'),
     'Y'^set>=1,
     'Y'^set<=size('X')]).

ctr_graph(
    inverse_set,
    ['X', 'Y'],
    2,
    ['PRODUCT'>>collection(x, y)],
    [in_set(y^index, x^set) #<=>in_set(x^index, y^set)],
    ['NARC'=size('X')*size('Y')]).

ctr_example(
    inverse_set,
    inverse_set(
        [[index-1, set-{2, 4}],
         [index-2, set-{4}],
         [index-3, set-{1}],
         [index-4, set-{4}]],
        [[index-1, set-{3}],
         [index-2, set-{1}],
         [index-3, set-{}],
         [index-4, set-{1, 2, 4}],
         [index-5, set-{}]])).

```


B.113 ith_pos_different_from_0

```

ctr_automaton(
    ith_pos_different_from_0,
    ith_pos_different_from_0).

ctr_date(ith_pos_different_from_0, ['20040811']).

ctr_origin(
    ith_pos_different_from_0,
    'Used for defining the automaton of %c.',
    [min_n]).

ctr_arguments(
    ith_pos_different_from_0,
    ['ITH'-int, 'POS'-dvar, 'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    ith_pos_different_from_0,
    ['ITH'>=1,
     'ITH'=<size('VARIABLES'),
     'POS'>='ITH',
     'POS'=<size('VARIABLES'),
     required('VARIABLES', var)]).

ctr_example(
    ith_pos_different_from_0,
    ith_pos_different_from_0(
        2,
        4,
        [[var-3], [var-0], [var-0], [var-8], [var-6]])).

ith_pos_different_from_0(A,B,C) :-
    ith_pos_different_from_0_signature(C,D),
    automaton(
        D,
        E,
        D,
        0..1,
        [source(s), sink(t)],
        [arc(s,0,s, (F#<A->[F+1,G+1])),
         arc(s,1,s, (F#<A->[F,G+1])),
         arc(s,$,t)],
        [F,G],
        [0,0],
        [A,B]).

```

```
ith_pos_different_from_0_signature([], []).
```

```
ith_pos_different_from_0_signature([[var-A]|B], [C|D]) :-  
    A#0#<=>C,  
    ith_pos_different_from_0_signature(B,D).
```

B.114 k_cut

```

ctr_date(k_cut, ['20030820', '20041230']).

ctr_origin(k_cut, 'E.~Althaus', []).

ctr_arguments(
    k_cut,
    ['K'-int, 'NODES'-collection(index-int, succ-svar)]).

ctr_restrictions(
    k_cut,
    ['K'>=1,
     'K'=<size('NODES'),
     required('NODES', [index, succ]),
     'NODES'^index>=1,
     'NODES'^index=<size('NODES'),
     distinct('NODES', index)]).

ctr_graph(
    k_cut,
    ['NODES'],
    2,
    ['CLIQUE'>>collection(nodes1, nodes2)],
    [#\/(nodes1^index=nodes2^index,
         in_set(nodes2^index, nodes1^succ))],
    ['NCC'>='K']).

ctr_example(
    k_cut,
    k_cut(
        3,
        [[index-1, succ-{}],
         [index-2, succ-{3, 5}],
         [index-3, succ-{5}],
         [index-4, succ-{}],
         [index-5, succ-{2, 3}]])).

```

B.115 lex2

```
ctr_predefined(lex2).  
  
ctr_date(lex2, ['20031008', '20040530']).  
  
ctr_origin(  
    lex2,  
    '\\cite{FlenerFrischHnichKiziltanMiguelPearsonWalsh02}',  
    []).  
  
ctr_synonyms(lex2, [double_lex, row_and_column_lex]).  
  
ctr_types(lex2, ['VECTOR'-collection(var-dvar)]).  
  
ctr_arguments(lex2, ['MATRIX'-collection(vec-'VECTOR')]).  
  
ctr_restrictions(  
    lex2,  
    [required('VECTOR', var),  
     required('MATRIX', vec),  
     same_size('MATRIX', vec)]).  
  
ctr_example(  
    lex2,  
    lex2(  
        [[vec-[[var-2], [var-2], [var-3]]],  
         [vec-[[var-2], [var-3], [var-1]]]]).  
    )
```

B.116 lex_alldifferent

```

ctr_date(lex_alldifferent, ['20030820', '20040530']).

ctr_origin(lex_alldifferent, 'J.~Pearson', []).

ctr_synonyms(lex_alldifferent, [lex_alldiff, lex_alldistinct]).

ctr_types(lex_alldifferent, ['VECTOR'-collection(var-dvar)]).

ctr_arguments(
    lex_alldifferent,
    ['VECTORS'-collection(vec-'VECTOR')]).

ctr_restrictions(
    lex_alldifferent,
    [required('VECTOR', var),
     required('VECTORS', vec),
     same_size('VECTORS', vec)]).

ctr_graph(
    lex_alldifferent,
    ['VECTORS'],
    2,
    ['CLIQUE' (<)>>collection(vectors1, vectors2)],
    [lex_different(vectors1^vec, vectors2^vec)],
    ['NARC'=size('VECTORS')*(size('VECTORS')-1)/2]).

ctr_example(
    lex_alldifferent,
    lex_alldifferent(
        [[vec-[[var-5], [var-2], [var-3]]],
         [vec-[[var-5], [var-2], [var-6]]],
         [vec-[[var-5], [var-3], [var-3]]]])).

```


B.117 lex_between

```

ctr_automaton(lex_between, lex_between) .

ctr_date(lex_between, ['20030820', '20040530']) .

ctr_origin(lex_between, '\\cite{BeldiceanuCarlsson02c}', []).

ctr_arguments(
  lex_between,
  ['LOWER_BOUND'-collection(var-int),
   'VECTOR'-collection(var-dvar),
   'UPPER_BOUND'-collection(var-int)]).

ctr_restrictions(
  lex_between,
  [required('LOWER_BOUND', var),
   required('VECTOR', var),
   required('UPPER_BOUND', var),
   size('LOWER_BOUND')=size('VECTOR'),
   size('UPPER_BOUND')=size('VECTOR'),
   lex_lesseq('LOWER_BOUND', 'VECTOR'),
   lex_lesseq('VECTOR', 'UPPER_BOUND')]).

ctr_example(
  lex_between,
  lex_between(
    [[var-5], [var-2], [var-3], [var-9]],
    [[var-5], [var-2], [var-6], [var-2]],
    [[var-5], [var-2], [var-6], [var-3]])).

lex_between(A,B,C) :-
  lex_between_signature(A,B,C,D),
  automaton(
    D,
    E,
    D,
    0..8,
    [source(s), node(a), node(b), sink(t)],
    [arc(s,4,s),
     arc(s,0,t),
     arc(s,$,t),
     arc(s,3,a),
     arc(s,1,b),
     arc(a,3,a),
     arc(a,4,a),

```

```

arc(a,5,a),
arc(a,0,t),
arc(a,1,t),
arc(a,2,t),
arc(a,$,t),
arc(b,1,b),
arc(b,4,b),
arc(b,7,b),
arc(b,0,t),
arc(b,3,t),
arc(b,6,t),
arc(b,$,t)],
[],
[],
[]).

```

```
lex_between_signature([],[],[],[]).
```

```

lex_between_signature(
  [[var-A]|B],
  [[var-C]|D],
  [[var-E]|F],
  [G|H]) :-
  I is A-1,
  J is A+1,
  K is E-1,
  L is E+1,
  ( A<E ->
    case(
      M-N,
      [C-G],
      [node(
        -1,
        M,
        [(inf..I)-6,
         (A..A)-3,
         (J..K)-0,
         (E..E)-1,
         (L..sup)-2]),
        node(0,N,[0..0]),
        node(1,N,[1..1]),
        node(2,N,[2..2]),
        node(3,N,[3..3]),
        node(6,N,[6..6])])
  ; A:=E ->
    case(

```

```

M-N,
[C-G],
[node(-1,M,[(inf..I)-6,(A..A)-4,(J..sup)-2]),
 node(2,N,[2..2]),
 node(4,N,[4..4]),
 node(6,N,[6..6])]
; A>E ->
case(
M-N,
[C-G],
[node(
-1,
M,
[(inf..K)-6,
(E..E)-7,
(L..I)-8,
(A..A)-5,
(J..sup)-2]),
node(2,N,[2..2]),
node(5,N,[5..5]),
node(6,N,[6..6]),
node(7,N,[7..7]),
node(8,N,[8..8])]
),
lex_between_signature(B,D,F,H) .

```

B.118 lex_chain_less

```

ctr_date(lex_chain_less, ['20030820', '20040530']).

ctr_origin(lex_chain_less, '\\cite{BeldiceanuCarlsson02c}', []).

ctr_usual_name(lex_chain_less, lex_chain).

ctr_types(lex_chain_less, ['VECTOR'-collection(var-dvar)]).

ctr_arguments(
  lex_chain_less,
  ['VECTORS'-collection(vec-'VECTOR')]).

ctr_restrictions(
  lex_chain_less,
  [required('VECTOR', var),
   required('VECTORS', vec),
   same_size('VECTORS', vec)]).

ctr_graph(
  lex_chain_less,
  ['VECTORS'],
  2,
  ['PATH'>>collection(vectors1, vectors2)],
  [lex_less(vectors1^vec, vectors2^vec)],
  ['NARC'=size('VECTORS')-1]).

ctr_example(
  lex_chain_less,
  lex_chain_less(
    [[vec-[[var-5], [var-2], [var-3], [var-9]]],
     [vec-[[var-5], [var-2], [var-6], [var-2]]],
     [vec-[[var-5], [var-2], [var-6], [var-3]]]])).

```

B.119 lex_chain_lesseq

```

ctr_date(lex_chain_lesseq, ['20030820', '20040530']).

ctr_origin(lex_chain_lesseq, '\\cite{BeldiceanuCarlsson02c}', []).

ctr_usual_name(lex_chain_lesseq, lex_chain).

ctr_types(lex_chain_lesseq, ['VECTOR'-collection(var-dvar)]).

ctr_arguments(
  lex_chain_lesseq,
  ['VECTORS'-collection(vec-'VECTOR')]).

ctr_restrictions(
  lex_chain_lesseq,
  [required('VECTOR', var),
   required('VECTORS', vec),
   same_size('VECTORS', vec)]).

ctr_graph(
  lex_chain_lesseq,
  ['VECTORS'],
  2,
  ['PATH'>>collection(vectors1, vectors2)],
  [lex_lesseq(vectors1^vec, vectors2^vec)],
  ['NARC'=size('VECTORS')-1]).

ctr_example(
  lex_chain_lesseq,
  lex_chain_lesseq(
    [[vec-[[var-5], [var-2], [var-3], [var-9]]],
     [vec-[[var-5], [var-2], [var-6], [var-2]]],
     [vec-[[var-5], [var-2], [var-6], [var-2]]]])).

```

B.120 `lex_different`

```

ctr_automaton(lex_different,lex_different).

ctr_date(lex_different,['20030820','20040530']).

ctr_origin(
    lex_different,
    'Used for defining %c.',
    [lex_alldifferent]).

ctr_arguments(
    lex_different,
    ['VECTOR1'-collection(var-dvar),
     'VECTOR2'-collection(var-dvar)]).

ctr_restrictions(
    lex_different,
    [required('VECTOR1',var),
     required('VECTOR2',var),
     size('VECTOR1')=size('VECTOR2')]).

ctr_graph(
    lex_different,
    ['VECTOR1','VECTOR2'],
    2,
    ['PRODUCT' (=)>>collection(vector1,vector2)],
    [vector1^var=\=vector2^var],
    ['NARC'>=1]).

ctr_example(
    lex_different,
    lex_different(
        [[var-5],[var-2],[var-7],[var-1]],
        [[var-5],[var-3],[var-7],[var-1]])).

lex_different(A,B) :-
    lex_different_signature(A,B,C),
    automaton(
        C,
        D,
        C,
        0..1,
        [source(s),sink(t)],
        [arc(s,1,s),arc(s,0,t)],
        []).

```

```
[],  
[]).
```

```
lex_different_signature([], [], []).
```

```
lex_different_signature([[var-A] | B], [[var-C] | D], [E | F]) :-  
    A#C#<=>E,  
    lex_different_signature(B, D, F).
```

B.121 `lex_greater`

```

ctr_automaton(lex_greater, lex_greater) .

ctr_date(lex_greater, ['20030820', '20040530']) .

ctr_origin(lex_greater, 'CHIP', []).

ctr_arguments(
    lex_greater,
    ['VECTOR1'-collection(var-dvar),
     'VECTOR2'-collection(var-dvar)]).

ctr_restrictions(
    lex_greater,
    [required('VECTOR1', var),
     required('VECTOR2', var),
     size('VECTOR1')=size('VECTOR2')]).

ctr_derived_collections(
    lex_greater,
    [col('DESTINATION'-collection(index-int, x-int, y-int),
        [item(index-0, x-0, y-0)]),
     col('COMPONENTS'-collection(index-int, x-dvar, y-dvar),
        [item(
            index-'VECTOR1'^key,
            x-'VECTOR1'^var,
            y-'VECTOR2'^var)])]).

ctr_graph(
    lex_greater,
    ['COMPONENTS', 'DESTINATION'],
    2,
    ['PRODUCT' ('PATH', 'VOID')>>collection(item1, item2)],
    [#\/(item2^index>0#\item1^x=item1^y,
        item2^index=0#\item1^x>item1^y)],
    ['PATH_FROM_TO' (index, 1, 0)=1]).

ctr_example(
    lex_greater,
    lex_greater(
        [[var-5], [var-2], [var-7], [var-1]],
        [[var-5], [var-2], [var-6], [var-2]])).

lex_greater(A, B) :-
    lex_greater_signature(A, B, C),

```



```
automaton(  
    C,  
    D,  
    C,  
    1..3,  
    [source(s),sink(t)],  
    [arc(s,2,s),arc(s,3,t)],  
    [],  
    [],  
    []).  
  
lex_greater_signature([],[],[]).  
  
lex_greater_signature([[var-A]|B],[[var-C]|D],[E|F]) :-  
    in(E,1..3),  
    A#<C#<=>E#=1,  
    A#=C#<=>E#=2,  
    A#>C#<=>E#=3,  
    lex_greater_signature(B,D,F).
```

B.122 `lex_greatereq`

```

ctr_automaton(lex_greatereq, lex_greatereq) .

ctr_date(lex_greatereq, ['20030820', '20040530']) .

ctr_origin(lex_greatereq, 'CHIP', []).

ctr_arguments(
    lex_greatereq,
    ['VECTOR1'-collection(var-dvar),
     'VECTOR2'-collection(var-dvar)]).

ctr_restrictions(
    lex_greatereq,
    [required('VECTOR1', var),
     required('VECTOR2', var),
     size('VECTOR1')=size('VECTOR2')]).

ctr_derived_collections(
    lex_greatereq,
    [col('DESTINATION'-collection(index-int, x-int, y-int),
        [item(index-0, x-0, y-0)]),
     col('COMPONENTS'-collection(index-int, x-dvar, y-dvar),
        [item(
            index-'VECTOR1'^key,
            x-'VECTOR1'^var,
            y-'VECTOR2'^var)])]).

ctr_graph(
    lex_greatereq,
    ['COMPONENTS', 'DESTINATION'],
    2,
    ['PRODUCT' ('PATH', 'VOID')>>collection(item1, item2)],
    [#\(\#\(/(item2^index>0#\(/item1^x=item1^y,
        #\(\#\(/(item1^index<size('VECTOR1'),
            item2^index=0),
            item1^x>item1^y)),
        #\(/(item1^index=size('VECTOR1')#\(/item2^index=0,
            item1^x>=item1^y))],
    ['PATH_FROM_TO' (index, 1, 0)=1]).

ctr_example(
    lex_greatereq,
    [lex_greatereq(
        [[var-5], [var-2], [var-8], [var-9]],

```

```

    [[var-5],[var-2],[var-6],[var-2]]),
lex_greatereq(
    [[var-5],[var-2],[var-3],[var-9]],
    [[var-5],[var-2],[var-3],[var-9]])).

```

```

lex_greatereq(A,B) :-
    lex_greatereq_signature(A,B,C),
    automaton(
        C,
        D,
        C,
        1..3,
        [source(s),sink(t)],
        [arc(s,2,s),arc(s,3,t),arc(s,$,t)],
        [],
        [],
        []).

```

```

lex_greatereq_signature([],[],[]).

```

```

lex_greatereq_signature([[var-A]|B],[[var-C]|D],[E|F]) :-
    in(E,1..3),
    A#<C#<=>E#=1,
    A#=C#<=>E#=2,
    A#>C#<=>E#=3,
    lex_greatereq_signature(B,D,F).

```

B.123 lex_less

```

ctr_automaton(lex_less, lex_less) .

ctr_date(lex_less, ['20030820', '20040530']) .

ctr_origin(lex_less, 'CHIP', []).

ctr_arguments(
    lex_less,
    ['VECTOR1'-collection(var-dvar),
     'VECTOR2'-collection(var-dvar)]).

ctr_restrictions(
    lex_less,
    [required('VECTOR1', var),
     required('VECTOR2', var),
     size('VECTOR1')=size('VECTOR2')]).

ctr_derived_collections(
    lex_less,
    [col('DESTINATION'-collection(index-int, x-int, y-int),
        [item(index-0, x-0, y-0)]),
     col('COMPONENTS'-collection(index-int, x-dvar, y-dvar),
        [item(
            index-'VECTOR1'^key,
            x-'VECTOR1'^var,
            y-'VECTOR2'^var)])]).

ctr_graph(
    lex_less,
    ['COMPONENTS', 'DESTINATION'],
    2,
    ['PRODUCT' ('PATH', 'VOID')>>collection(item1, item2)],
    [#\/(item2^index>0#\item1^x=item1^y,
        item2^index=0#\item1^x<item1^y)],
    ['PATH_FROM_TO' (index, 1, 0)=1]).

ctr_example(
    lex_less,
    lex_less(
        [[var-5], [var-2], [var-3], [var-9]],
        [[var-5], [var-2], [var-6], [var-2]])).

lex_less(A, B) :-
    lex_less_signature(A, B, C),

```

```
automaton(  
    C,  
    D,  
    C,  
    1..3,  
    [source(s),sink(t)],  
    [arc(s,2,s),arc(s,1,t)],  
    [],  
    [],  
    []).  
  
lex_less_signature([],[],[]).  
  
lex_less_signature([[var-A]|B],[[var-C]|D],[E|F]) :-  
    in(E,1..3),  
    A#<C#<=>E#=1,  
    A#=C#<=>E#=2,  
    A#>C#<=>E#=3,  
    lex_less_signature(B,D,F).
```

B.124 lex_lesseq

```

ctr_automaton(lex_lesseq, lex_lesseq) .

ctr_date(lex_lesseq, ['20030820', '20040530']) .

ctr_origin(lex_lesseq, 'CHIP', []).

ctr_arguments(
    lex_lesseq,
    ['VECTOR1'-collection(var-dvar),
     'VECTOR2'-collection(var-dvar)]).

ctr_restrictions(
    lex_lesseq,
    [required('VECTOR1', var),
     required('VECTOR2', var),
     size('VECTOR1')=size('VECTOR2')]).

ctr_derived_collections(
    lex_lesseq,
    [col('DESTINATION'-collection(index-int, x-int, y-int),
        [item(index-0, x-0, y-0)]),
     col('COMPONENTS'-collection(index-int, x-dvar, y-dvar),
        [item(
            index-'VECTOR1'^key,
            x-'VECTOR1'^var,
            y-'VECTOR2'^var)])]).

ctr_graph(
    lex_lesseq,
    ['COMPONENTS', 'DESTINATION'],
    2,
    ['PRODUCT' ('PATH', 'VOID')>>collection(item1, item2)],
    [#\/(#\/(item2^index>0#\item1^x=item1^y,
        #/\(#\/(item1^index<size('VECTOR1'),
            item2^index=0),
            item1^x<item1^y)),
     #/\(item1^index=size('VECTOR1')#\item2^index=0,
        item1^x=<item1^y)],
    ['PATH_FROM_TO' (index, 1, 0)=1]).

ctr_example(
    lex_lesseq,
    [lex_lesseq(
        [[var-5], [var-2], [var-3], [var-1]],

```

```

    [[var-5],[var-2],[var-6],[var-2]]),
lex_lesseq(
    [[var-5],[var-2],[var-3],[var-9]],
    [[var-5],[var-2],[var-3],[var-9]])).

```

```

lex_lesseq(A,B) :-
    lex_lesseq_signature(A,B,C),
    automaton(
        C,
        D,
        C,
        1..3,
        [source(s),sink(t)],
        [arc(s,2,s),arc(s,1,t),arc(s,$,t)],
        [],
        [],
        []).

```

```

lex_lesseq_signature([],[],[]).

```

```

lex_lesseq_signature([[var-A]|B],[[var-C]|D],[E|F]) :-
    in(E,1..3),
    A#<C#<=>E#=1,
    A#=C#<=>E#=2,
    A#>C#<=>E#=3,
    lex_lesseq_signature(B,D,F).

```

B.125 link_set_to_booleans

```

ctr_date(link_set_to_booleans, ['20030820']).

ctr_origin(
  link_set_to_booleans,
  'Inspired by %c.',
  [domain_constraint]).

ctr_arguments(
  link_set_to_booleans,
  ['SVAR'-svar, 'BOOLEANS'-collection(bool-dvar, val-int)]).

ctr_restrictions(
  link_set_to_booleans,
  [required('BOOLEANS', [bool, val]),
   'BOOLEANS'^bool>=0,
   'BOOLEANS'^bool<=1,
   distinct('BOOLEANS', val)]).

ctr_derived_collections(
  link_set_to_booleans,
  [col('SET'-collection(one-int, setvar-svar),
      [item(one-1, setvar-'SVAR')])]).

ctr_graph(
  link_set_to_booleans,
  ['SET', 'BOOLEANS'],
  2,
  ['PRODUCT'>>collection(set, booleans)],
  [booleans^bool=set^one#<=>in_set(booleans^val, set^setvar)],
  ['NARC'=size('BOOLEANS')]).

ctr_example(
  link_set_to_booleans,
  link_set_to_booleans(
    {1, 3, 4},
    [[bool-0, val-0],
     [bool-1, val-1],
     [bool-0, val-2],
     [bool-1, val-3],
     [bool-1, val-4],
     [bool-0, val-5]])).

```


B.126 longest_change

```

ctr_automaton(longest_change, longest_change) .

ctr_date(longest_change, ['20000128', '20030820', '20040530']) .

ctr_origin(longest_change, 'Derived from %c.', [change]) .

ctr_arguments(
    longest_change,
    ['SIZE'-dvar, 'VARIABLES'-collection(var-dvar), 'CTR'-atom]) .

ctr_restrictions(
    longest_change,
    ['SIZE'>=0,
     'SIZE'<size('VARIABLES'),
     required('VARIABLES', var),
     in_list('CTR', [=,=\=, <, >=, >, =<])]) .

ctr_graph(
    longest_change,
    ['VARIABLES'],
    2,
    ['PATH'>>collection(variables1, variables2)],
    ['CTR'(variables1^var, variables2^var)],
    ['MAX_NCC'='SIZE']) .

ctr_example(
    longest_change,
    longest_change(
        4,
        [[var-8],
         [var-8],
         [var-3],
         [var-4],
         [var-1],
         [var-1],
         [var-5],
         [var-5],
         [var-2]],
        =\=)) .

longest_change(A, B, C) :-
    longest_change_signature(B, D, C),
    automaton(
        D,

```

```

E,
D,
0..1,
[source(s),sink(t)],
[arc(s,1,s,[F,G+1]),
 arc(s,0,s,[max(F,G),1]),
 arc(s,$,t,[max(F,G),G])],
[F,G],
[0,1],
[A,H]).

```

```
longest_change_signature([],[],A).
```

```
longest_change_signature([A],[],B) :-
!.
```

```
longest_change_signature([[var-A],[var-B]|C],[D|E],=) :-
!,
A#=B#<=>D,
longest_change_signature([[var-B]|C],E,=).
```

```
longest_change_signature([[var-A],[var-B]|C],[D|E],=\=) :-
!,
A#\=B#<=>D,
longest_change_signature([[var-B]|C],E,=\=).
```

```
longest_change_signature([[var-A],[var-B]|C],[D|E],<) :-
!,
A#<B#<=>D,
longest_change_signature([[var-B]|C],E,<).
```

```
longest_change_signature([[var-A],[var-B]|C],[D|E],>=) :-
!,
A#>=B#<=>D,
longest_change_signature([[var-B]|C],E,>=).
```

```
longest_change_signature([[var-A],[var-B]|C],[D|E],>) :-
!,
A#>B#<=>D,
longest_change_signature([[var-B]|C],E,>).
```

```
longest_change_signature([[var-A],[var-B]|C],[D|E],=<) :-
!,
A#=<B#<=>D,
longest_change_signature([[var-B]|C],E,=<).
```

B.127 map

```

ctr_date(map, ['20000128', '20030820']).

ctr_origin(map, 'Inspired by \\cite{SedgewickFlajolet96}', []).

ctr_arguments(
  map,
  ['NBCYCLE'-dvar,
   'NBTREE'-dvar,
   'NODES'-collection(index-int, succ-dvar)].

ctr_restrictions(
  map,
  ['NBCYCLE'>=0,
   'NBTREE'>=0,
   required('NODES', [index, succ]),
   'NODES'^index>=1,
   'NODES'^index<size('NODES'),
   distinct('NODES', index),
   'NODES'^succ>=1,
   'NODES'^succ<size('NODES')]).

ctr_graph(
  map,
  ['NODES'],
  2,
  ['CLIQUE'>>collection(nodes1, nodes2)],
  [nodes1^succ=nodes2^index],
  ['NCC'='NBCYCLE', 'NTREE'='NBTREE']).

ctr_example(
  map,
  map(2,
    3,
    [[index-1, succ-5],
     [index-2, succ-9],
     [index-3, succ-8],
     [index-4, succ-2],
     [index-5, succ-9],
     [index-6, succ-2],
     [index-7, succ-9],
     [index-8, succ-8],
     [index-9, succ-1]])).

```

B.128 max_index

```

ctr_automaton(max_index,max_index) .

ctr_date(max_index,['20030820','20040530','20041230']) .

ctr_origin(max_index,'N.~Beldiceanu',[ ]) .

ctr_arguments(
    max_index,
    ['MAX_INDEX'-dvar,
     'VARIABLES'-collection(index-int,var-dvar)]) .

ctr_restrictions(
    max_index,
    [size('VARIABLES')>0,
     'MAX_INDEX'>=0,
     'MAX_INDEX'=<size('VARIABLES'),
     required('VARIABLES',[index,var]),
     'VARIABLES'^index>=1,
     'VARIABLES'^index=<size('VARIABLES'),
     distinct('VARIABLES',index)]) .

ctr_graph(
    max_index,
    ['VARIABLES'],
    2,
    ['CLIQUE'>>collection(variables1,variables2)],
    [#\/(variables1^key=variables2^key,
        variables1^var>variables2^var)],
    ['ORDER'(0,0,index)='MAX_INDEX']) .

ctr_example(
    max_index,
    max_index(
        3,
        [[index-1,var-3],
         [index-2,var-2],
         [index-3,var-7],
         [index-4,var-2],
         [index-5,var-6]]) .

max_index(A,B) :-
    length(B,C),
    length(D,C),
    domain(D,0,0),

```

```
max_index_signature(B,E,D),
automaton(
  E,
  F,
  D,
  0..0,
  [source(s),sink(t)],
  [arc(s,0,s,(F#=<G->[G,H,I+1];F#>G->[F,I+1,I+1])),
  arc(s,$,t)],
  [G,H,I],
  [-1000000,0,0],
  [J,A,K]).
```

```
max_index_signature([],[],[]).
```

```
max_index_signature([[index-A,var-B]|C],[B|D],[0|E]) :-
  max_index_signature(C,D,E).
```

B.129 max_n

```

ctr_date(max_n, ['20000128', '20030820', '20041230']).

ctr_origin(max_n, '\\cite{Beldiceanu01}', []).

ctr_arguments(
    max_n,
    ['MAX'-dvar, 'RANK'-int, 'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    max_n,
    [size('VARIABLES')>0,
     'RANK'>=0,
     'RANK'<size('VARIABLES'),
     required('VARIABLES', var)]).

ctr_graph(
    max_n,
    ['VARIABLES'],
    2,
    ['CLIQUE'>>collection(variables1, variables2)],
    [#\/(variables1^key=variables2^key,
        variables1^var>variables2^var)],
    ['ORDER' ('RANK', 'MININT', var)='MAX']).

ctr_example(
    max_n,
    max_n(6, 1, [[var-3], [var-1], [var-7], [var-1], [var-6]])).

```

B.130 max_nvalue

```
ctr_date(max_nvalue, ['20000128', '20030820']).

ctr_origin(max_nvalue, 'Derived from %c.', [nvalue]).

ctr_arguments(
    max_nvalue,
    ['MAX'-dvar, 'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    max_nvalue,
    ['MAX'>=1,
     'MAX'=<size('VARIABLES'),
     required('VARIABLES', var)]).

ctr_graph(
    max_nvalue,
    ['VARIABLES'],
    2,
    ['CLIQUE'>>collection(variables1, variables2)],
    [variables1^var=variables2^var],
    ['MAX_NSCC'='MAX']).

ctr_example(
    max_nvalue,
    max_nvalue(
        3,
        [[var-9],
         [var-1],
         [var-7],
         [var-1],
         [var-1],
         [var-6],
         [var-7],
         [var-7],
         [var-4],
         [var-9]])).
```

B.131 max_size_set_of_consecutive_var

```

ctr_date(
  max_size_set_of_consecutive_var,
  ['20030820', '20040530']).

ctr_origin(max_size_set_of_consecutive_var, 'N.~Beldiceanu', []).

ctr_arguments(
  max_size_set_of_consecutive_var,
  ['MAX'-dvar, 'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
  max_size_set_of_consecutive_var,
  ['MAX'>=1,
   'MAX'=<size('VARIABLES'),
   required('VARIABLES', var)]).

ctr_graph(
  max_size_set_of_consecutive_var,
  ['VARIABLES'],
  2,
  ['CLIQUE'>>collection(variables1, variables2)],
  [abs(variables1^var-variables2^var)=<1],
  ['MAX_NSCC'='MAX']).

ctr_example(
  max_size_set_of_consecutive_var,
  max_size_set_of_consecutive_var(
    6,
    [[var-3],
     [var-1],
     [var-3],
     [var-7],
     [var-4],
     [var-1],
     [var-2],
     [var-8],
     [var-7],
     [var-6]])).

```


B.132 maximum

```

ctr_automaton(maximum,maximum) .

ctr_date(maximum,['20000128','20030820','20040530','20041230']) .

ctr_origin(maximum,'CHIP',[]) .

ctr_arguments(
    maximum,
    ['MAX'-dvar,'VARIABLES'-collection(var-dvar)]) .

ctr_restrictions(
    maximum,
    [size('VARIABLES')>0,required('VARIABLES',var)]) .

ctr_graph(
    maximum,
    ['VARIABLES'],
    2,
    ['CLIQUE'>>collection(variables1,variables2)],
    [#\/(variables1^key=variables2^key,
        variables1^var>variables2^var)],
    ['ORDER'(0,'MININT',var)='MAX'] .

ctr_example(
    maximum,
    maximum(7,[var-3],[var-2],[var-7],[var-2],[var-6])) .

maximum(A,B) :-
    maximum_signature(B,C,A),
    automaton(
        C,
        D,
        C,
        0..2,
        [source(s),node(e),sink(t)],
        [arc(s,0,s),
         arc(s,1,e),
         arc(e,1,e),
         arc(e,0,e),
         arc(e,$,t)],
        [],
        [],
        []).

```

```
maximum_signature([], [], A).
```

```
maximum_signature([[var-A] | B], [C | D], E) :-  
    in(C, 0..2),  
    E#>A#<=>C#=0,  
    E#=A#<=>C#=1,  
    E#<A#<=>C#=2,  
    maximum_signature(B, D, E).
```

B.133 maximum_modulo

```
ctr_date(maximum_modulo, ['20000128', '20030820', '20041230']).
ctr_origin(maximum_modulo, 'Derived from %c.', [maximum]).
ctr_arguments(
    maximum_modulo,
    ['MAX'-dvar, 'VARIABLES'-collection(var-dvar), 'M'-int]).
ctr_restrictions(
    maximum_modulo,
    [size('VARIABLES')>0, 'M'>0, required('VARIABLES', var)]).
ctr_graph(
    maximum_modulo,
    ['VARIABLES'],
    2,
    ['CLIQUE'>>collection(variables1, variables2)],
    [#\/(variables1^key=variables2^key,
        variables1^var mod 'M'>variables2^var mod 'M')],
    ['ORDER' (0, 'MININT', var)='MAX']).
ctr_example(
    maximum_modulo,
    maximum_modulo(
        5,
        [[var-9], [var-1], [var-7], [var-6], [var-5]],
        3)).
```

B.134 min_index

```

ctr_automaton(min_index,min_index) .

ctr_date(min_index,['20030820','20040530','20041230']) .

ctr_origin(min_index,'N.~Beldiceanu',[ ]) .

ctr_arguments(
    min_index,
    ['MIN_INDEX'-dvar,
     'VARIABLES'-collection(index-int,var-dvar)]) .

ctr_restrictions(
    min_index,
    [size('VARIABLES')>0,
     'MIN_INDEX'>=0,
     'MIN_INDEX'=<size('VARIABLES'),
     required('VARIABLES',[index,var]),
     'VARIABLES'^index>=1,
     'VARIABLES'^index=<size('VARIABLES'),
     distinct('VARIABLES',index)]) .

ctr_graph(
    min_index,
    ['VARIABLES'],
    2,
    ['CLIQUE'>>collection(variables1,variables2)],
    [#\/(variables1^key=variables2^key,
        variables1^var<variables2^var)],
    ['ORDER'(0,0,index)='MIN_INDEX']) .

ctr_example(
    min_index,
    [min_index(
        2,
        [[index-1,var-3],
         [index-2,var-2],
         [index-3,var-7],
         [index-4,var-2],
         [index-5,var-6]]),
     min_index(
        4,
        [[index-1,var-3],
         [index-2,var-2],
         [index-3,var-7],

```

```
[index-4,var-2],
[index-5,var-6]]]).
```

```
min_index(A,B) :-
    length(B,C),
    length(D,C),
    domain(D,0,0),
    min_index_signature(B,E,D),
    automaton(
        E,
        F,
        D,
        0..0,
        [source(s),sink(t)],
        [arc(s,0,s,(F#>=G->[G,H,I+1];F#<G->[F,I+1,I+1])),
        arc(s,$,t)],
        [G,H,I],
        [1000000,0,0],
        [J,A,K]).
```

```
min_index_signature([],[],[]).
```

```
min_index_signature([[index-A,var-B]|C],[B|D],[0|E]) :-
    min_index_signature(C,D,E).
```

B.135 min_n

```

ctr_date(min_n, ['20000128', '20030820', '20040530', '20041230']).
ctr_origin(min_n, '\\cite{Beldiceanu01}', []).
ctr_arguments(
  min_n,
  ['MIN'-dvar, 'RANK'-int, 'VARIABLES'-collection(var-dvar)]).
ctr_restrictions(
  min_n,
  [size('VARIABLES')>0,
   'RANK'>=0,
   'RANK'<size('VARIABLES'),
   required('VARIABLES', var)]).
ctr_graph(
  min_n,
  ['VARIABLES'],
  2,
  ['CLIQUE'>>collection(variables1, variables2)],
  [#\/(variables1^key=variables2^key,
      variables1^var<variables2^var)],
  ['ORDER' ('RANK', 'MAXINT', var)='MIN']).
ctr_example(
  min_n,
  min_n(3, 1, [[var-3], [var-1], [var-7], [var-1], [var-6]])).

```

B.136 min_nvalue

```

ctr_date(min_nvalue, ['20000128', '20030820']).

ctr_origin(min_nvalue, 'N.~Beldiceanu', []).

ctr_arguments(
    min_nvalue,
    ['MIN'-dvar, 'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    min_nvalue,
    ['MIN'>=1,
     'MIN'=<size('VARIABLES'),
     required('VARIABLES', var)]).

ctr_graph(
    min_nvalue,
    ['VARIABLES'],
    2,
    ['CLIQUE'>>collection(variables1, variables2)],
    [variables1^var=variables2^var],
    ['MIN_NSCC'='MIN']).

ctr_example(
    min_nvalue,
    min_nvalue(
        2,
        [[var-9],
         [var-1],
         [var-7],
         [var-1],
         [var-1],
         [var-7],
         [var-7],
         [var-7],
         [var-7],
         [var-9]])).

```

B.137 min_size_set_of_consecutive_var

```

ctr_date(
  min_size_set_of_consecutive_var,
  ['20030820', '20040530']).

ctr_origin(min_size_set_of_consecutive_var, 'N.~Beldiceanu', []).

ctr_arguments(
  min_size_set_of_consecutive_var,
  ['MIN'-dvar, 'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
  min_size_set_of_consecutive_var,
  ['MIN'>=1,
   'MIN'=<size('VARIABLES'),
   required('VARIABLES', var)]).

ctr_graph(
  min_size_set_of_consecutive_var,
  ['VARIABLES'],
  2,
  ['CLIQUE'>>collection(variables1, variables2)],
  [abs(variables1^var-variables2^var)=<1],
  ['MIN_NSCC'='MIN']).

ctr_example(
  min_size_set_of_consecutive_var,
  min_size_set_of_consecutive_var(
    4,
    [[var-3],
     [var-1],
     [var-3],
     [var-7],
     [var-4],
     [var-1],
     [var-2],
     [var-8],
     [var-7],
     [var-6]])).

```


B.138 minimum

```

ctr_automaton(minimum,minimum) .

ctr_date(minimum, ['20000128', '20030820', '20040530', '20041230'] ) .

ctr_origin(minimum, 'CHIP', []) .

ctr_arguments(
    minimum,
    ['MIN'-dvar, 'VARIABLES'-collection(var-dvar)] ) .

ctr_restrictions(
    minimum,
    [size('VARIABLES')>0, required('VARIABLES', var)] ) .

ctr_graph(
    minimum,
    ['VARIABLES'],
    2,
    ['CLIQUE'>>collection(variables1,variables2)],
    [#\/(variables1^key=variables2^key,
        variables1^var<variables2^var)],
    ['ORDER'(0, 'MAXINT', var)='MIN'] ) .

ctr_example(
    minimum,
    minimum(2, [[var-3], [var-2], [var-7], [var-2], [var-6]]) ) .

minimum(A,B) :-
    minimum_signature(B,C,A),
    automaton(
        C,
        D,
        C,
        0..2,
        [source(s), node(e), sink(t)],
        [arc(s,0,s),
         arc(s,1,e),
         arc(e,1,e),
         arc(e,0,e),
         arc(e,$,t)],
        [],
        [],
        []) .

```

```
minimum_signature([], [], A).
```

```
minimum_signature([[var-A] | B], [C | D], E) :-  
    in(C, 0..2),  
    E#<A#<=>C#=0,  
    E#=A#<=>C#=1,  
    E#>A#<=>C#=2,  
    minimum_signature(B, D, E).
```

B.139 minimum_except_0

```

ctr_automaton(minimum_except_0,minimum_except_0).

ctr_date(minimum_except_0,['20030820','20040530','20041230']).

ctr_origin(minimum_except_0,'Derived from %c.',[minimum]).

ctr_arguments(
    minimum_except_0,
    ['MIN'-dvar,'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    minimum_except_0,
    [size('VARIABLES')>0,
     required('VARIABLES',var),
     'VARIABLES'^var>=0]).

ctr_graph(
    minimum_except_0,
    ['VARIABLES'],
    2,
    ['CLIQUE'>>collection(variables1,variables2)],
    [variables1^var=\=0,
     variables2^var=\=0,
     #\/(variables1^key=variables2^key,
          variables1^var<variables2^var)],
    ['ORDER'(0,'MAXINT',var)='MIN']]).

ctr_example(
    minimum_except_0,
    [minimum_except_0(
        3,
        [[var-3],[var-7],[var-6],[var-7],[var-4],[var-7]]),
     minimum_except_0(
        2,
        [[var-3],[var-2],[var-0],[var-7],[var-2],[var-6]]),
     minimum_except_0(
        1000000,
        [[var-0],[var-0],[var-0],[var-0],[var-0],[var-0]])]).

minimum_except_0(A,B) :-
    minimum_except_0_signature(B,C,A),
    automaton(
        C,
        D,

```

```

C,
0..4,
[source(s),node(j),node(k),sink(t)],
[arc(s,0,s),
 arc(s,3,s),
 arc(s,2,j),
 arc(s,1,k),
 arc(j,0,j),
 arc(j,1,j),
 arc(j,2,j),
 arc(j,3,j),
 arc(j,$,t),
 arc(k,1,k),
 arc(k,$,t)],
[],
[],
[]).

```

```
minimum_except_0_signature([],[],A).
```

```

minimum_except_0_signature([[var-A]|B],[C|D],E) :-
  in(C,0..4),
  F=1000000,
  A#=0#\E#\=F#<=>C#=0,
  A#=0#\E#=F#<=>C#=1,
  A#\=0#\E#=A#<=>C#=2,
  A#\=0#\E#<A#<=>C#=3,
  A#\=0#\E#>A#<=>C#=4,
  minimum_except_0_signature(B,D,E).

```

B.140 minimum_greater_than

```

ctr_automaton(minimum_greater_than,minimum_greater_than).

ctr_date(minimum_greater_than,['20030820']).

ctr_origin(minimum_greater_than,'N.~Beldiceanu',[ ]).

ctr_arguments(
    minimum_greater_than,
    ['VAR1'-dvar,'VAR2'-dvar,'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    minimum_greater_than,
    [size('VARIABLES')>0,required('VARIABLES',var)]).

ctr_derived_collections(
    minimum_greater_than,
    [col('ITEM'-collection(var-dvar),[item(var-'VAR2')])]).

ctr_graph(
    minimum_greater_than,
    ['ITEM','VARIABLES'],
    2,
    ['PRODUCT'>>collection(item,variables)],
    [item^var<variables^var],
    ['NARC'>0],
    ['SUCC'>>[source,variables]],
    [minimum('VAR1',variables)]).

ctr_example(
    minimum_greater_than,
    minimum_greater_than(
        5,
        3,
        [[var-8],[var-5],[var-3],[var-8]]).

minimum_greater_than(A,B,C) :-
    minimum_greater_than_signature(C,D,A,B),
    automaton(
        D,
        E,
        D,
        0..5,
        [source(s),node(e),sink(t)],
        [arc(s,0,s),

```

```

arc(s,1,s),
arc(s,2,s),
arc(s,5,s),
arc(s,4,e),
arc(e,0,e),
arc(e,1,e),
arc(e,2,e),
arc(e,4,e),
arc(e,5,e),
arc(e,$,t)],
[],
[],
[]).

```

```

minimum_greater_than_signature([],[],A,B).

```

```

minimum_greater_than_signature([[var-A]|B],[C|D],E,F) :-
    in(C,0..5),
    A#<E#\A#=<F#<=>C#=0,
    A#=E#\A#=<F#<=>C#=1,
    A#>E#\A#=<F#<=>C#=2,
    A#<E#\A#>F#<=>C#=3,
    A#=E#\A#>F#<=>C#=4,
    A#>E#\A#>F#<=>C#=5,
    minimum_greater_than_signature(B,D,E,F).

```

B.141 minimum_modulo

```

ctr_date(minimum_modulo, ['20000128', '20030820', '20041230']).
ctr_origin(minimum_modulo, 'Derived from %c.', [minimum]).
ctr_arguments(
    minimum_modulo,
    ['MIN'-dvar, 'VARIABLES'-collection(var-dvar), 'M'-int]).
ctr_restrictions(
    minimum_modulo,
    [size('VARIABLES')>0, 'M'>0, required('VARIABLES', var)]).
ctr_graph(
    minimum_modulo,
    ['VARIABLES'],
    2,
    ['CLIQUE'>>collection(variables1, variables2)],
    [#\/(variables1^key=variables2^key,
        variables1^var mod 'M'<variables2^var mod 'M')],
    ['ORDER' (0, 'MAXINT', var)='MIN']]).
ctr_example(
    minimum_modulo,
    [minimum_modulo(
        6,
        [[var-9], [var-1], [var-7], [var-6], [var-5]],
        3),
    minimum_modulo(
        9,
        [[var-9], [var-1], [var-7], [var-6], [var-5]],
        3)]).

```

B.142 `minimum_weight_alldifferent`

```
ctr_date(minimum_weight_alldifferent, ['20030820', '20040530']).
```

```
ctr_origin(
  minimum_weight_alldifferent,
  '\\cite{FocacciLodiMilano99}',
  []).
```

```
ctr_synonyms(
  minimum_weight_alldifferent,
  [minimum_weight_alldiff,
   minimum_weight_alldistinct,
   min_weight_alldiff,
   min_weight_alldifferent,
   min_weight_alldistinct]).
```

```
ctr_arguments(
  minimum_weight_alldifferent,
  ['VARIABLES'-collection(var-dvar),
   'MATRIX'-collection(i-int, j-int, c-int),
   'COST'-dvar]).
```

```
ctr_restrictions(
  minimum_weight_alldifferent,
  [size('VARIABLES')>0,
   required('VARIABLES', var),
   'VARIABLES'^var>=1,
   'VARIABLES'^var<=size('VARIABLES'),
   required('MATRIX', [i, j, c]),
   increasing_seq('MATRIX', [i, j]),
   'MATRIX'^i>=1,
   'MATRIX'^i<=size('VARIABLES'),
   'MATRIX'^j>=1,
   'MATRIX'^j<=size('VARIABLES'),
   size('MATRIX')==size('VARIABLES')*size('VARIABLES')]).
```

```
ctr_graph(
  minimum_weight_alldifferent,
  ['VARIABLES'],
  2,
  ['CLIQUE'>>collection(variables1, variables2)],
  [variables1^var=variables2^key],
  ['NTREE'=0,
   =('SUM_WEIGHT_ARC' (
     ^(@('MATRIX',
```



```

        + ((variables1^key-1)*size('VARIABLES'),
          variables1^var)),
      c)),
    'COST')]).

ctr_example(
  minimum_weight_alldifferent,
  minimum_weight_alldifferent(
    [[var-2],[var-3],[var-1],[var-4]],
    [[i-1,j-1,c-4],
     [i-1,j-2,c-1],
     [i-1,j-3,c-7],
     [i-1,j-4,c-0],
     [i-2,j-1,c-1],
     [i-2,j-2,c-0],
     [i-2,j-3,c-8],
     [i-2,j-4,c-2],
     [i-3,j-1,c-3],
     [i-3,j-2,c-2],
     [i-3,j-3,c-1],
     [i-3,j-4,c-6],
     [i-4,j-1,c-0],
     [i-4,j-2,c-0],
     [i-4,j-3,c-6],
     [i-4,j-4,c-5]],
    17)).

```

B.143 nclass

```

ctr_date(nclass, ['20000128', '20030820']).

ctr_origin(nclass, 'Derived from %c.', [nvalue]).

ctr_types(nclass, ['VALUES'-collection(val-int)]).

ctr_arguments(
    nclass,
    ['NCLASS'-dvar,
     'VARIABLES'-collection(var-dvar),
     'PARTITIONS'-collection(p-'VALUES')]).

ctr_restrictions(
    nclass,
    [required('VALUES', val),
     distinct('VALUES', val),
     'NCLASS' >= 0,
     'NCLASS' = <min(size('VARIABLES'), size('PARTITIONS'))],
    required('VARIABLES', var),
    required('PARTITIONS', p),
    size('PARTITIONS') >= 2]).

ctr_graph(
    nclass,
    ['VARIABLES'],
    2,
    ['CLIQUE' >> collection(variables1, variables2)],
    [in_same_partition(
        variables1^var,
        variables2^var,
        'PARTITIONS')],
    ['NSCC' = 'NCLASS']).

ctr_example(
    nclass,
    nclass(
        2,
        [[var-3], [var-2], [var-7], [var-2], [var-6]],
        [[p-[val-1], [val-3]]],
        [p-[val-4]],
        [p-[val-2], [val-6]]])).

```

B.144 nequivalence

```

ctr_date(nequivalence, ['20000128', '20030820']).

ctr_origin(nequivalence, 'Derived from %c.', [nvalue]).

ctr_arguments(
    nequivalence,
    ['NEQUIV'-dvar, 'M'-int, 'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    nequivalence,
    ['NEQUIV' >= min(1, size('VARIABLES')),
     'NEQUIV' <= min('M', size('VARIABLES')),
     'M' > 0,
     required('VARIABLES', var)]).

ctr_graph(
    nequivalence,
    ['VARIABLES'],
    2,
    ['CLIQUE' >> collection(variables1, variables2)],
    [variables1^var mod 'M' = variables2^var mod 'M'],
    ['NSCC' = 'NEQUIV']).

ctr_example(
    nequivalence,
    nequivalence(
        2,
        3,
        [[var-3],
         [var-2],
         [var-5],
         [var-6],
         [var-15],
         [var-3],
         [var-3]])).

```

B.145 next_element

```

ctr_automaton(next_element,next_element).

ctr_date(next_element,['20030820','20040530']).

ctr_origin(next_element,'N.~Beldiceanu',[ ]).

ctr_arguments(
    next_element,
    ['THRESHOLD'-dvar,
     'INDEX'-dvar,
     'TABLE'-collection(index-int,value-dvar),
     'VAL'-dvar]).

ctr_restrictions(
    next_element,
    ['INDEX'>=1,
     'INDEX'=<size('TABLE'),
     required('TABLE',[index,value]),
     'TABLE'^index>=1,
     'TABLE'^index=<size('TABLE'),
     distinct('TABLE',index)]).

ctr_derived_collections(
    next_element,
    [col('ITEM'-collection(index-dvar,value-dvar),
        [item(index-'THRESHOLD',value-'VAL')])]).

ctr_graph(
    next_element,
    ['ITEM','TABLE'],
    2,
    ['PRODUCT'>>>collection(item,table)],
    [item^index<table^index,item^value=table^value],
    ['NARC'>0],
    [>>('SUCC',
        [source,
         -(variables,
            col('VARIABLES'-collection(var-dvar),
                [item(var-'TABLE'^index)])])]),
    [minimum('INDEX',variables)]).

ctr_example(
    next_element,
    next_element(

```

```

2,
3,
[[index-1,value-1],
 [index-2,value-8],
 [index-3,value-9],
 [index-4,value-5],
 [index-5,value-9]],
9)).

```

```

next_element(A,B,C,D) :-
  next_element_signature(C,E,A,B,D),
  automaton(
    E,
    F,
    E,
    0..11,
    [source(s),node(e),sink(t)],
    [arc(s,0,s),
     arc(s,1,s),
     arc(s,2,s),
     arc(s,3,s),
     arc(s,4,s),
     arc(s,5,s),
     arc(s,7,s),
     arc(s,9,s),
     arc(s,10,s),
     arc(s,11,s),
     arc(s,8,e),
     arc(e,0,e),
     arc(e,1,e),
     arc(e,2,e),
     arc(e,3,e),
     arc(e,4,e),
     arc(e,5,e),
     arc(e,7,e),
     arc(e,8,e),
     arc(e,9,e),
     arc(e,10,e),
     arc(e,11,e),
     arc(e,$,t)],
    [],
    [],
    []).

```

```

next_element_signature([],[],A,B,C).

```

```

next_element_signature([ [index-A,value-B] | C], [D|E], F, G, H) :-
    in(D, 0..11),
    A#=<F#/\A#<G#/\B#=H#<=>D#=0,
    A#=<F#/\A#<G#/\B#\=H#<=>D#=1,
    A#=<F#/\A#=G#/\B#=H#<=>D#=2,
    A#=<F#/\A#=G#/\B#\=H#<=>D#=3,
    A#=<F#/\A#>G#/\B#=H#<=>D#=4,
    A#=<F#/\A#>G#/\B#\=H#<=>D#=5,
    A#>F#/\A#<G#/\B#=H#<=>D#=6,
    A#>F#/\A#<G#/\B#\=H#<=>D#=7,
    A#>F#/\A#=G#/\B#=H#<=>D#=8,
    A#>F#/\A#=G#/\B#\=H#<=>D#=9,
    A#>F#/\A#>G#/\B#=H#<=>D#=10,
    A#>F#/\A#>G#/\B#\=H#<=>D#=11,
    next_element_signature(C, E, F, G, H) .

```

B.146 next_greater_element

```

ctr_date(next_greater_element, ['20030820', '20040530']).

ctr_origin(next_greater_element, 'M.~Carlsson', []).

ctr_arguments(
  next_greater_element,
  ['VAR1'-dvar, 'VAR2'-dvar, 'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
  next_greater_element,
  [size('VARIABLES')>0, required('VARIABLES', var)]).

ctr_derived_collections(
  next_greater_element,
  [col('V'-collection(var-dvar), [item(var-'VAR1')])]).

ctr_graph(
  next_greater_element,
  ['VARIABLES'],
  2,
  ['PATH'>>collection(variables1, variables2)],
  [variables1^var<variables2^var],
  ['NARC'=size('VARIABLES')-1]).

ctr_graph(
  next_greater_element,
  ['V', 'VARIABLES'],
  2,
  ['PRODUCT'>>collection(v, variables)],
  [v^var<variables^var],
  ['NARC'>0],
  ['SUCC'>>[source, variables]],
  [minimum('VAR2', variables)]).

ctr_example(
  next_greater_element,
  next_greater_element(
    7,
    8,
    [[var-3], [var-5], [var-8], [var-9]])).

```

B.147 ninterval

```

ctr_date(ninterval, ['20030820', '20040530']).

ctr_origin(ninterval, 'Derived from %c.', [nvalue]).

ctr_arguments(
    ninterval,
    ['NVAL'-dvar,
     'VARIABLES'-collection(var-dvar),
     'SIZE_INTERVAL'-int]).

ctr_restrictions(
    ninterval,
    ['NVAL'>=min(1, size('VARIABLES')),
     'NVAL'=<size('VARIABLES'),
     required('VARIABLES', var),
     'SIZE_INTERVAL'>0]).

ctr_graph(
    ninterval,
    ['VARIABLES'],
    2,
    ['CLIQUE'>>collection(variables1, variables2)],
    [(variables1^var/'SIZE_INTERVAL',
     variables2^var/'SIZE_INTERVAL')],
    ['NSCC'='NVAL']).

ctr_example(
    ninterval,
    ninterval(2, [[var-3], [var-1], [var-9], [var-1], [var-9]], 4)).

```


B.148 no_peak

```

ctr_automaton(no_peak,no_peak) .

ctr_date(no_peak,['20031101','20040530']) .

ctr_origin(no_peak,'Derived from %c.',[peak]) .

ctr_arguments(no_peak,['VARIABLES'-collection(var-dvar)]) .

ctr_restrictions(
    no_peak,
    [size('VARIABLES')>0,required('VARIABLES',var)]) .

ctr_example(
    no_peak,
    no_peak([[var-1],[var-1],[var-4],[var-8],[var-8]]) .

no_peak(A) :-
    no_peak_signature(A,B),
    automaton(
        B,
        C,
        B,
        0..2,
        [source(s),node(i),sink(t)],
        [arc(s,0,s),
         arc(s,1,s),
         arc(s,2,i),
         arc(s,$,t),
         arc(i,1,i),
         arc(i,2,i),
         arc(i,$,t)],
        [],
        [],
        []).

no_peak_signature([],[]).

no_peak_signature([A],[]).

no_peak_signature([[var-A],[var-B]|C],[D|E]) :-
    in(D,0..2),
    A#<B#<=>D#=0,
    A#=B#<=>D#=1,
    A#>B#<=>D#=2,

```

```
no_peak_signature([[var-B] | C], E) .
```

B.149 no_valley

```

ctr_automaton(no_valley,no_valley) .

ctr_date(no_valley,['20031101','20040530']) .

ctr_origin(no_valley,'Derived from %c.',[valley]) .

ctr_arguments(no_valley,['VARIABLES'-collection(var-dvar)]) .

ctr_restrictions(
    no_valley,
    [size('VARIABLES')>0,required('VARIABLES',var)]) .

ctr_example(
    no_valley,
    no_valley(
        [[var-1],[var-1],[var-4],[var-8],[var-8],[var-2]]) .

no_valley(A) :-
    no_valley_signature(A,B),
    automaton(
        B,
        C,
        B,
        0..2,
        [source(s),node(i),sink(t)],
        [arc(s,0,s),
         arc(s,1,s),
         arc(s,2,i),
         arc(s,$,t),
         arc(i,1,i),
         arc(i,2,i),
         arc(i,$,t)],
        [],
        [],
        []).

no_valley_signature([],[]).

no_valley_signature([A],[]).

no_valley_signature([[var-A],[var-B]|C],[D|E]) :-
    in(D,0..2),
    A#<B#<=>D#=0,
    A#=B#<=>D#=1,

```

```
A#>B#<=>D#=2,  
no_valley_signature([[var-B]|C],E).
```

B.150 not_all_equal

```

ctr_automaton(not_all_equal,not_all_equal) .

ctr_date(not_all_equal,['20030820','20040530','20040726']) .

ctr_origin(not_all_equal,'CHIP',[ ]) .

ctr_arguments(not_all_equal,['VARIABLES'-collection(var-dvar)]) .

ctr_restrictions(
  not_all_equal,
  [required('VARIABLES',var),size('VARIABLES')>1]) .

ctr_graph(
  not_all_equal,
  ['VARIABLES'],
  2,
  ['CLIQUE'>>collection(variables1,variables2)],
  [variables1^var=variables2^var],
  ['NSCC'>1]) .

ctr_example(
  not_all_equal,
  not_all_equal([[var-3],[var-1],[var-3],[var-3],[var-3]]) .

not_all_equal(A) :-
  length(A,B),
  B>1,
  not_all_equal_signature(A,C),
  automaton(
    C,
    D,
    C,
    0..1,
    [source(s),sink(t)],
    [arc(s,1,s),arc(s,0,t)],
    [],
    [],
    []).

not_all_equal_signature([],[]) .

not_all_equal_signature([A],[ ]) .

not_all_equal_signature([[var-A],[var-B]|C],[D|E]) :-

```

```
A#=B#<=>D,  
not_all_equal_signature([[var-B]|C],E).
```

B.151 not_in

```

ctr_automaton(not_in,not_in) .

ctr_date(not_in,['20030820','20040530']) .

ctr_origin(not_in,'Derived from %c.',[in]) .

ctr_arguments(not_in,['VAR'-dvar,'VALUES'-collection(val-int)]) .

ctr_restrictions(
    not_in,
    [required('VALUES',val),distinct('VALUES',val)]) .

ctr_derived_collections(
    not_in,
    [col('VARIABLES'-collection(var-dvar),[item(var-'VAR')])]) .

ctr_graph(
    not_in,
    ['VARIABLES','VALUES'],
    2,
    ['PRODUCT'>>collection(variables,values)],
    [variables^var=values^val],
    ['NARC'=0]) .

ctr_example(not_in,not_in(2,[[val-1],[val-3]])) .

not_in(A,B) :-
    not_in_signature(B,C,A),
    automaton(
        C,
        D,
        C,
        0..1,
        [source(s),sink(t)],
        [arc(s,0,s),arc(s,$,t)],
        [],
        [],
        []).

not_in_signature([],[],A) .

not_in_signature([[val-A]|B],[C|D],E) :-
    E#=A#<=>C,
    not_in_signature(B,D,E) .

```


B.152 npair

```
ctr_date(npair, ['20030820']).

ctr_origin(npair, 'Derived from %c.', [nvalue]).

ctr_arguments(
    npair,
    ['NVAL'-dvar, 'PAIRS'-collection(x-dvar, y-dvar)]).

ctr_restrictions(
    npair,
    ['NVAL' >= min(1, size('PAIRS')),
     'NVAL' <= size('PAIRS'),
     required('PAIRS', [x, y])]).

ctr_graph(
    npair,
    ['PAIRS'],
    2,
    ['CLIQUE' >> collection(pairs1, pairs2)],
    [pairs1^x=pairs2^x, pairs1^y=pairs2^y],
    ['NSCC'='NVAL']).

ctr_example(
    npair,
    npair(
        2,
        [[x-3, y-1], [x-1, y-5], [x-3, y-1], [x-3, y-1], [x-1, y-5]])).
```

B.153 nset_of_consecutive_values

```

ctr_date(nset_of_consecutive_values, ['20030820', '20040530']).

ctr_origin(nset_of_consecutive_values, 'N.~Beldiceanu', []).

ctr_arguments(
  nset_of_consecutive_values,
  ['N'-dvar, 'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
  nset_of_consecutive_values,
  ['N'>=1, 'N'=<size('VARIABLES'), required('VARIABLES', var)]).

ctr_graph(
  nset_of_consecutive_values,
  ['VARIABLES'],
  2,
  ['CLIQUE'>>collection(variables1, variables2)],
  [abs(variables1^var-variables2^var)=<1],
  ['NSCC'='N']).

ctr_example(
  nset_of_consecutive_values,
  nset_of_consecutive_values(
    2,
    [[var-3],
     [var-1],
     [var-7],
     [var-1],
     [var-1],
     [var-2],
     [var-8]])).

```

B.154 nvalue

```
ctr_date(nvalue, ['20000128', '20030820', '20040530']).  
ctr_origin(nvalue, '\\cite{PachetRoy99}', []).  
ctr_synonyms(nvalue, [cardinality_on_attributes_values]).  
ctr_arguments(  
    nvalue,  
    ['NVAL'-dvar, 'VARIABLES'-collection(var-dvar)]).  
ctr_restrictions(  
    nvalue,  
    ['NVAL' >=min(1, size('VARIABLES')),  
     'NVAL' <=size('VARIABLES'),  
     required('VARIABLES', var)]).  
ctr_graph(  
    nvalue,  
    ['VARIABLES'],  
    2,  
    ['CLIQUE' >>>collection(variables1, variables2)],  
    [variables1^var=variables2^var],  
    ['NSCC'='NVAL']).  
ctr_example(  
    nvalue,  
    nvalue(4, [[var-3], [var-1], [var-7], [var-1], [var-6]])).
```

B.155 nvalue_on_intersection

```

ctr_date(nvalue_on_intersection, ['20040530']).

ctr_origin(
    nvalue_on_intersection,
    'Derived from %c and %c.',
    [common, nvalue]).

ctr_arguments(
    nvalue_on_intersection,
    ['NVAL'-dvar,
     'VARIABLES1'-collection(var-dvar),
     'VARIABLES2'-collection(var-dvar)]).

ctr_restrictions(
    nvalue_on_intersection,
    ['NVAL'>=0,
     'NVAL'=<size('VARIABLES1'),
     'NVAL'=<size('VARIABLES2'),
     required('VARIABLES1', var),
     required('VARIABLES2', var)]).

ctr_graph(
    nvalue_on_intersection,
    ['VARIABLES1', 'VARIABLES2'],
    2,
    ['PRODUCT'>>collection(variables1, variables2)],
    [variables1^var=variables2^var],
    ['NCC'='NVAL']).

ctr_example(
    nvalue_on_intersection,
    nvalue_on_intersection(
        2,
        [[var-1], [var-9], [var-1], [var-5]],
        [[var-2], [var-1], [var-9], [var-9], [var-6], [var-9]])).

```

B.156 nvalues

```

ctr_date(nvalues, ['20030820']).

ctr_origin(nvalues, 'Inspired by %c and %c.', [nvalue, count]).

ctr_arguments(
    nvalues,
    ['VARIABLES'-collection(var-dvar),
     'RELOP'-atom,
     'LIMIT'-dvar]).

ctr_restrictions(
    nvalues,
    [required('VARIABLES', var),
     in_list('RELOP', [=,=\=, <, >=, >, =<])]).

ctr_graph(
    nvalues,
    ['VARIABLES'],
    2,
    ['CLIQUE'>>collection(variables1, variables2)],
    [variables1^var=variables2^var],
    ['RELOP' ('NSCC', 'LIMIT')]).

ctr_example(
    nvalues,
    nvalues(
        [[var-4], [var-5], [var-5], [var-4], [var-1], [var-5]],
        =,
        3)).

```

B.157 nvalues_except_0

```

ctr_date(nvalues_except_0, ['20030820']).

ctr_origin(nvalues_except_0, 'Derived from %c.', [nvalues]).

ctr_arguments(
    nvalues_except_0,
    ['VARIABLES'-collection(var-dvar),
     'RELOP'-atom,
     'LIMIT'-dvar]).

ctr_restrictions(
    nvalues_except_0,
    [required('VARIABLES', var),
     in_list('RELOP', [=,=\=, <, >=, >, =<])]).

ctr_graph(
    nvalues_except_0,
    ['VARIABLES'],
    2,
    ['CLIQUE'>>collection(variables1, variables2)],
    [variables1^var=\=0, variables1^var=variables2^var],
    ['RELOP' ('NSCC', 'LIMIT')]).

ctr_example(
    nvalues_except_0,
    nvalues_except_0(
        [[var-4], [var-5], [var-5], [var-4], [var-0], [var-1]],
        =,
        3)).

```

B.158 one_tree

```

ctr_date(one_tree, ['20031001', '20040530']).

ctr_origin(
  one_tree,
  'Inspired by \\cite{GentProsserSmithWei03}',
  []).

ctr_arguments(
  one_tree,
  [-( 'NODES',
      collection(
        id-atom,
        index-int,
        type-int,
        father-dvar,
        depth1-dvar,
        depth2-dvar))]).

ctr_restrictions(
  one_tree,
  [required('NODES', [id, index, type, father, depth1, depth2]),
   'NODES'^index>=1,
   'NODES'^index<=size('NODES'),
   distinct('NODES', index),
   in_list('NODES', type, [2, 3, 6]),
   'NODES'^father>=1,
   'NODES'^father<=size('NODES'),
   'NODES'^depth1>=0,
   'NODES'^depth1<=size('NODES'),
   'NODES'^depth2>=0,
   'NODES'^depth2<=size('NODES')]).

ctr_graph(
  one_tree,
  ['NODES'],
  2,
  ['CLIQUE'>>collection(nodes1, nodes2)],
  [#\/(#\/(nodes1^index=nodes2^index,
           nodes1^father=nodes1^index),
   #\/(#\/(#\/(nodes1^index=\=nodes2^index,
              nodes1^father=nodes2^index),
   #\/(#\/(nodes1^type mod 2=0,
          nodes1^depth1>nodes2^depth1),
   #\/(nodes1^type mod 2>0,

```

```

        nodes1^depth1=nodes2^depth1))),
#\/(\/(nodes1^type mod 3=0,
        nodes1^depth2>nodes2^depth2),
#\/(nodes1^type mod 3>0,
        nodes1^depth2=nodes2^depth2))),
['MAX_NSCC'=<1,'NCC'=1,'NVERTEX'=size('NODES')]).

```

```

ctr_example(
  one_tree,
  one_tree(
    [id-x,index-1,type-2,father-6,depth1-2,depth2-2],
    [id-x,index-2,type-2,father-2,depth1-1,depth2-0],
    [id-x,index-3,type-3,father-6,depth1-1,depth2-3],
    [id-x,index-4,type-3,father-5,depth1-2,depth2-4],
    [id-x,index-5,type-3,father-1,depth1-2,depth2-3],
    [id-x,index-6,type-3,father-7,depth1-1,depth2-2],
    [id-x,index-7,type-3,father-2,depth1-1,depth2-1],
    [id-g,index-8,type-2,father-1,depth1-3,depth2-2],
    [id-a,index-9,type-6,father-4,depth1-3,depth2-5],
    [id-f,index-10,type-6,father-7,depth1-2,depth2-2],
    [id-b,index-11,type-3,father-4,depth1-2,depth2-5],
    [id-c,index-12,type-3,father-5,depth1-2,depth2-4],
    [id-e,index-13,type-3,father-3,depth1-1,depth2-4],
    [id-d,index-14,type-3,father-3,depth1-1,depth2-4])).

```


B.159 orchard

```

ctr_date(orchard, ['20000128', '20030820']).

ctr_origin(orchard, '\\cite{Jackson1821}', []).

ctr_arguments(
  orchard,
  ['NROW'-dvar, 'TREES'-collection(index-int, x-dvar, y-dvar)]).

ctr_restrictions(
  orchard,
  ['NROW' >= 0,
   'TREES'^index >= 1,
   'TREES'^index <= size('TREES'),
   required('TREES', [index, x, y]),
   distinct('TREES', index),
   'TREES'^x >= 0,
   'TREES'^y >= 0]).

ctr_graph(
  orchard,
  ['TREES'],
  3,
  ['CLIQUE' (<)>>collection(trees1, trees2, trees3)],
  [= (+ (+ (trees1^x*trees2^y - trees1^x*trees3^y,
            trees1^y*trees3^x - trees1^y*trees2^x),
            trees2^x*trees3^y - trees2^y*trees3^x),
      0)],
  ['NARC' = 'NROW'] ).

ctr_example(
  orchard,
  orchard(
    10,
    [[index-1, x-0, y-0],
     [index-2, x-4, y-0],
     [index-3, x-8, y-0],
     [index-4, x-2, y-4],
     [index-5, x-4, y-4],
     [index-6, x-6, y-4],
     [index-7, x-0, y-8],
     [index-8, x-4, y-8],
     [index-9, x-8, y-8]])).

```

B.160 orth_link_ori_siz_end

```

ctr_date(orth_link_ori_siz_end, ['20030820']).

ctr_origin(
  orth_link_ori_siz_end,
  'Used by several constraints between orthotopes',
  []).

ctr_arguments(
  orth_link_ori_siz_end,
  ['ORTHOTOPE'-collection(ori-dvar, siz-dvar, end-dvar)]).

ctr_restrictions(
  orth_link_ori_siz_end,
  [size('ORTHOTOPE')>0,
   require_at_least(2, 'ORTHOTOPE', [ori, siz, end]),
   'ORTHOTOPE'^siz>=0]).

ctr_graph(
  orth_link_ori_siz_end,
  ['ORTHOTOPE'],
  1,
  ['SELF'>>collection(orthotope)],
  [orthotope^ori+orthotope^siz=orthotope^end],
  ['NARC'=size('ORTHOTOPE')]).

ctr_example(
  orth_link_ori_siz_end,
  orth_link_ori_siz_end(
    [[ori-2, siz-2, end-4], [ori-1, siz-3, end-4]])).

```

B.161 orth_on_the_ground

```

ctr_date(orth_on_the_ground, ['20030820', '20040726']).

ctr_origin(
    orth_on_the_ground,
    'Used for defining %c.',
    [place_in_pyramid]).

ctr_arguments(
    orth_on_the_ground,
    ['ORTHOTOPE'-collection(ori-dvar, siz-dvar, end-dvar),
     'VERTICAL_DIM'-int]).

ctr_restrictions(
    orth_on_the_ground,
    [size('ORTHOTOPE')>0,
     require_at_least(2, 'ORTHOTOPE', [ori, siz, end]),
     'ORTHOTOPE'^siz>=0,
     'VERTICAL_DIM'>=1,
     'VERTICAL_DIM'=<size('ORTHOTOPE'),
     orth_link_ori_siz_end('ORTHOTOPE')]).

ctr_graph(
    orth_on_the_ground,
    ['ORTHOTOPE'],
    1,
    ['SELF'>>collection(orthotope)],
    [orthotope^key='VERTICAL_DIM', orthotope^ori=1],
    ['NARC'=1]).

ctr_example(
    orth_on_the_ground,
    orth_on_the_ground(
        [[ori-1, siz-2, end-3], [ori-2, siz-3, end-5]],
        1)).

```

B.162 orth_on_top_of_orth

```

ctr_date(orth_on_top_of_orth, ['20030820', '20040726']).

ctr_origin(
    orth_on_top_of_orth,
    'Used for defining %c.',
    [place_in_pyramid]).

ctr_types(
    orth_on_top_of_orth,
    ['ORTHOTOPE'-collection(ori-dvar, siz-dvar, end-dvar)]).

ctr_arguments(
    orth_on_top_of_orth,
    ['ORTHOTOPE1'-'ORTHOTOPE',
     'ORTHOTOPE2'-'ORTHOTOPE',
     'VERTICAL_DIM'-int]).

ctr_restrictions(
    orth_on_top_of_orth,
    [size('ORTHOTOPE')>0,
     require_at_least(2, 'ORTHOTOPE', [ori, siz, end]),
     'ORTHOTOPE'^siz>=0,
     size('ORTHOTOPE1')=size('ORTHOTOPE2'),
     'VERTICAL_DIM'>=1,
     'VERTICAL_DIM'=<size('ORTHOTOPE1'),
     orth_link_ori_siz_end('ORTHOTOPE1'),
     orth_link_ori_siz_end('ORTHOTOPE2')]).

ctr_graph(
    orth_on_top_of_orth,
    ['ORTHOTOPE1', 'ORTHOTOPE2'],
    2,
    ['PRODUCT' (=)>>collection(orthotope1, orthotope2)],
    [orthotope1^key=\='VERTICAL_DIM',
     orthotope2^ori=<orthotope1^ori,
     orthotope1^end=<orthotope2^end],
    ['NARC'=size('ORTHOTOPE1')-1]).

ctr_graph(
    orth_on_top_of_orth,
    ['ORTHOTOPE1', 'ORTHOTOPE2'],
    2,
    ['PRODUCT' (=)>>collection(orthotope1, orthotope2)],
    [orthotope1^key='VERTICAL_DIM',

```

```
    orthotope1^ori=orthotope2^end],  
    ['NARC'=1]).  
  
ctr_example(  
    orth_on_top_of_orth,  
    orth_on_top_of_orth(  
        [[ori-5,siz-2,end-7],[ori-3,siz-3,end-6]],  
        [[ori-3,siz-5,end-8],[ori-1,siz-2,end-3]],  
        2)).
```

B.163 `orths_are_connected`

```

ctr_date(orths_are_connected, ['20000128', '20030820']).

ctr_origin(orths_are_connected, 'N.~Beldiceanu', []).

ctr_types(
    orths_are_connected,
    ['ORTHOTOPE'-collection(ori-dvar, siz-dvar, end-dvar)]).

ctr_arguments(
    orths_are_connected,
    ['ORTHOTOPES'-collection(orth-'ORTHOTOPE')]).

ctr_restrictions(
    orths_are_connected,
    [size('ORTHOTOPE')>0,
     require_at_least(2, 'ORTHOTOPE', [ori, siz, end]),
     'ORTHOTOPE'^siz>0,
     required('ORTHOTOPES', orth),
     same_size('ORTHOTOPES', orth)].

ctr_graph(
    orths_are_connected,
    ['ORTHOTOPES'],
    1,
    ['SELF'>>collection(orthotopes)],
    [orth_link_ori_siz_end(orthotopes^orth)],
    ['NARC'=size('ORTHOTOPES')]).

ctr_graph(
    orths_are_connected,
    ['ORTHOTOPES'],
    2,
    ['CLIQUE' (=|=)>>collection(orthotopes1, orthotopes2)],
    [two_orth_are_in_contact(
        orthotopes1^orth,
        orthotopes2^orth)],
    ['NVERTEX'=size('ORTHOTOPES'), 'NCC'=1]).

ctr_example(
    orths_are_connected,
    orths_are_connected(
        [[orth-[ [ori-2, siz-4, end-6], [ori-2, siz-2, end-4]]],
         [orth-[ [ori-1, siz-2, end-3], [ori-4, siz-3, end-7]]],
         [orth-[ [ori-7, siz-4, end-11], [ori-1, siz-2, end-3]]],

```

[orth-[[ori-6,siz-2,end-8],[ori-3,siz-2,end-5]]]) .

B.164 path_from_to

```
ctr_date(path_from_to, ['20030820', '20040530']).
```

```
ctr_origin(
  path_from_to,
  '\\cite{AlthausBockmayrElfKasperJungerMehlhorn02}',
  []).
```

```
ctr_usual_name(path_from_to, path).
```

```
ctr_arguments(
  path_from_to,
  ['FROM'-int,
   'TO'-int,
   'NODES'-collection(index-int, succ-svar)]).
```

```
ctr_restrictions(
  path_from_to,
  ['FROM'>=1,
   'FROM'=<size('NODES'),
   'TO'>=1,
   'TO'=<size('NODES'),
   required('NODES', [index, succ]),
   'NODES'^index>=1,
   'NODES'^index=<size('NODES'),
   distinct('NODES', index)]).
```

```
ctr_graph(
  path_from_to,
  ['NODES'],
  2,
  ['CLIQUE'>>collection(nodes1, nodes2)],
  [in_set(nodes2^index, nodes1^succ)],
  ['PATH_FROM_TO'(index, 'FROM', 'TO')=1]).
```

```
ctr_example(
  path_from_to,
  path_from_to(
    4,
    3,
    [[index-1, succ-{}],
     [index-2, succ-{}],
     [index-3, succ-{5}],
     [index-4, succ-{5}],
     [index-5, succ-{2, 3}]]).
```


B.165 pattern

```

ctr_predefined(pattern) .

ctr_date(pattern, ['20031008']) .

ctr_origin(pattern, '\\cite{BourdaisGalini erPesant03}', []).

ctr_types(pattern, ['PATTERN'-collection(var-int)]).

ctr_arguments(
    pattern,
    ['VARIABLES'-collection(var-dvar),
     'PATTERNS'-collection(pat-'PATTERN')]).

ctr_restrictions(
    pattern,
    [required('PATTERN', var),
     change(0, 'PATTERN', =),
     required('VARIABLES', var),
     required('PATTERNS', pat),
     same_size('PATTERNS', pat)]).

ctr_example(
    pattern,
    pattern(
        [[var-1],
         [var-1],
         [var-2],
         [var-2],
         [var-2],
         [var-1],
         [var-3],
         [var-3]],
        [[pat-[[var-1], [var-2], [var-1]]],
         [pat-[[var-1], [var-2], [var-3]]],
         [pat-[[var-2], [var-1], [var-3]]]])).

```

B.166 peak

```

ctr_automaton(peak,peak) .

ctr_date(peak,['20040530']) .

ctr_origin(peak,'Derived from %c.',[inflexion]) .

ctr_arguments(peak,['N'-dvar,'VARIABLES'-collection(var-dvar)]) .

ctr_restrictions(
    peak,
    ['N'>=0,
     2*'N'=<max(size('VARIABLES')-1,0),
     required('VARIABLES',var)]) .

ctr_example(
    peak,
    peak(
        2,
        [[var-1],
         [var-1],
         [var-4],
         [var-8],
         [var-6],
         [var-2],
         [var-7],
         [var-1]])) .

peak(A,B) :-
    peak_signature(B,C),
    automaton(
        C,
        D,
        C,
        0..2,
        [source(s),node(u),sink(t)],
        [arc(s,0,s),
         arc(s,1,s),
         arc(s,2,u),
         arc(s,$,t),
         arc(u,0,s,[E+1]),
         arc(u,1,u),
         arc(u,2,u),
         arc(u,$,t)],
        [E],

```

```
[0],  
[A]).  
  
peak_signature([], []).  
  
peak_signature([A], []).  
  
peak_signature([[var-A], [var-B] | C], [D|E]) :-  
    in(D, 0..2),  
    A#>B#<=>D#=0,  
    A#=B#<=>D#=1,  
    A#<B#<=>D#=2,  
    peak_signature([[var-B] | C], E).
```

B.167 period

```
ctr_predefined(period) .

ctr_date(period, ['20000128', '20030820', '20040530']) .

ctr_origin(period, 'N.~Beldiceanu', []).

ctr_arguments(
    period,
    ['PERIOD'-dvar,
     'VARIABLES'-collection(var-dvar),
     'CTR'-atom]) .

ctr_restrictions(
    period,
    ['PERIOD'>=1,
     'PERIOD'=<size('VARIABLES'),
     required('VARIABLES', var),
     in_list('CTR', [=,=\=,<,>=,>,<=])]) .

ctr_example(
    period,
    period(
        3,
        [[var-1],
         [var-1],
         [var-4],
         [var-1],
         [var-1],
         [var-4],
         [var-1],
         [var-1]],
    =)).
```

B.168 period_except_0

```

ctr_predefined(period_except_0).

ctr_date(period_except_0, ['20030820', '20040530']).

ctr_origin(period_except_0, 'Derived from %c.', [period]).

ctr_arguments(
    period_except_0,
    ['PERIOD'-dvar,
     'VARIABLES'-collection(var-dvar),
     'CTR'-atom]).

ctr_restrictions(
    period_except_0,
    ['PERIOD'>=1,
     'PERIOD'=<size('VARIABLES'),
     required('VARIABLES', var),
     in_list('CTR', [=,=\=,<,>=,>,<=])]).

ctr_example(
    period_except_0,
    period_except_0(
        3,
        [[var-1],
         [var-1],
         [var-4],
         [var-1],
         [var-1],
         [var-0],
         [var-1],
         [var-1]],
        =)).

```

B.169 place_in_pyramid

```
ctr_date(place_in_pyramid, ['20000128', '20030820', '20041230']).
```

```
ctr_origin(place_in_pyramid, 'N.~Beldiceanu', []).
```

```
ctr_types(
    place_in_pyramid,
    ['ORTHOTOPE'-collection(ori-dvar, siz-dvar, end-dvar)]).
```

```
ctr_arguments(
    place_in_pyramid,
    ['ORTHOTOPES'-collection(orth-'ORTHOTOPE'),
     'VERTICAL_DIM'-int]).
```

```
ctr_restrictions(
    place_in_pyramid,
    [size('ORTHOTOPE')>0,
     require_at_least(2, 'ORTHOTOPE', [ori, siz, end]),
     'ORTHOTOPE'^siz>=0,
     same_size('ORTHOTOPES', orth),
     'VERTICAL_DIM'>=1,
     diffn('ORTHOTOPES')]).
```

```
ctr_graph(
    place_in_pyramid,
    ['ORTHOTOPES'],
    2,
    ['CLIQUE'>>collection(orthotopes1, orthotopes2)],
    [#\/(#\ (orthotopes1^key=orthotopes2^key,
            orth_on_the_ground(
                orthotopes1^orth,
                'VERTICAL_DIM')),
     #\/(orthotopes1^key=\=orthotopes2^key,
            orth_on_top_of_orth(
                orthotopes1^orth,
                orthotopes2^orth,
                'VERTICAL_DIM')))],
    ['NARC'=size('ORTHOTOPES')]).
```

```
ctr_example(
    place_in_pyramid,
    place_in_pyramid(
        [[orth-[ori-1, siz-3, end-4], [ori-1, siz-2, end-3]],
         [orth-[ori-1, siz-2, end-3], [ori-3, siz-3, end-6]],
         [orth-[ori-5, siz-6, end-11], [ori-1, siz-2, end-3]]],
```

```
[orth-[[ori-5,siz-2,end-7],[ori-3,siz-2,end-5]]],  
[orth-[[ori-8,siz-3,end-11],[ori-3,siz-2,end-5]]],  
[orth-[[ori-8,siz-2,end-10],[ori-5,siz-2,end-7]]],  
2)).
```


B.170 polyomino

```

ctr_date(polyomino, ['20000128', '20030820']).

ctr_origin(polyomino, 'Inspired by \\cite{Golomb65}.'. , []).

ctr_arguments(
    polyomino,
    [-( 'CELLS',
        collection(
            index-int,
            right-dvar,
            left-dvar,
            up-dvar,
            down-dvar))]).

ctr_restrictions(
    polyomino,
    ['CELLS'^index>=1,
     'CELLS'^index<=size('CELLS'),
     size('CELLS')>=1,
     required('CELLS', [index, right, left, up, down]),
     distinct('CELLS', index),
     'CELLS'^right>=0,
     'CELLS'^right<=size('CELLS'),
     'CELLS'^left>=0,
     'CELLS'^left<=size('CELLS'),
     'CELLS'^up>=0,
     'CELLS'^up<=size('CELLS'),
     'CELLS'^down>=0,
     'CELLS'^down<=size('CELLS')]).

ctr_graph(
    polyomino,
    ['CELLS'],
    2,
    ['CLIQUE' (=\\=)>>collection(cells1, cells2)],
    [#\\/(#\\/(#\\/(#\\/(cells1^right=cells2^index,
                        cells2^left=cells1^index),
                        #\\/(cells1^left=cells2^index,
                        cells2^right=cells1^index)),
                        #\\/(cells1^up=cells2^index,
                        cells2^down=cells1^index)),
     cells1^down=cells2^index#\\/(cells2^up=cells1^index)],
    ['NVERTEX'=size('CELLS'), 'NCC'=1]).

```

```
ctr_example(  
  polyomino,  
  polyomino(  
    [[index-1, right-0, left-0, up-2, down-0],  
     [index-2, right-3, left-0, up-0, down-1],  
     [index-3, right-0, left-2, up-4, down-0],  
     [index-4, right-5, left-0, up-0, down-3],  
     [index-5, right-0, left-4, up-0, down-0]])).
```

B.171 product_ctr

```
ctr_date(product_ctr, ['20030820']).

ctr_origin(product_ctr, 'Arithmetic constraint.', []).

ctr_arguments(
    product_ctr,
    ['VARIABLES'-collection(var-dvar), 'CTR'-atom, 'VAR'-dvar]).

ctr_restrictions(
    product_ctr,
    [required('VARIABLES', var),
     in_list('CTR', [=,=\=, <, >=, >, =<])]).

ctr_graph(
    product_ctr,
    ['VARIABLES'],
    1,
    ['SELF'>>collection(variables)],
    ['TRUE'],
    ['CTR' ('PRODUCT' ('VARIABLES', var), 'VAR')]).

ctr_example(
    product_ctr,
    product_ctr([[var-2], [var-1], [var-4]], =, 8)).
```

B.172 range_ctr

```
ctr_date(range_ctr, ['20030820']).

ctr_origin(range_ctr, 'Arithmetic constraint.', []).

ctr_arguments(
    range_ctr,
    ['VARIABLES'-collection(var-dvar), 'CTR'-atom, 'VAR'-dvar]).

ctr_restrictions(
    range_ctr,
    [required('VARIABLES', var),
     in_list('CTR', [=,=\=, <, >=, >, =<])]).

ctr_graph(
    range_ctr,
    ['VARIABLES'],
    1,
    ['SELF'>>collection(variables)],
    ['TRUE'],
    ['CTR' ('RANGE' ('VARIABLES', var), 'VAR')]).

ctr_example(range_ctr, range_ctr([[var-1], [var-9], [var-4]], =, 8)).
```

B.173 relaxed_sliding_sum

```

ctr_date(relaxed_sliding_sum, ['20000128', '20030820']).

ctr_origin(relaxed_sliding_sum, 'CHIP', []).

ctr_arguments(
    relaxed_sliding_sum,
    ['ATLEAST'-int,
     'ATMOST'-int,
     'LOW'-int,
     'UP'-int,
     'SEQ'-int,
     'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    relaxed_sliding_sum,
    ['ATLEAST'>=0,
     'ATMOST'>='ATLEAST',
     'ATMOST'=<size('VARIABLES')-'SEQ'+1,
     'UP'>='LOW',
     'SEQ'>0,
     'SEQ'=<size('VARIABLES'),
     required('VARIABLES', var)]).

ctr_graph(
    relaxed_sliding_sum,
    ['VARIABLES'],
    'SEQ',
    ['PATH'>>collection],
    [sum_ctr(collection, >=, 'LOW'), sum_ctr(collection, <=, 'UP')],
    ['NARC'>='ATLEAST', 'NARC'=<'ATMOST']).

ctr_example(
    relaxed_sliding_sum,
    relaxed_sliding_sum(
        3,
        4,
        3,
        7,
        4,
        [[var-2],
         [var-4],
         [var-2],
         [var-0],
         [var-0],

```

```
[var-3],  
[var-4]))).
```

B.174 same

```

ctr_date(same, ['20000128', '20030820', '20040530']).

ctr_origin(same, 'N.~Beldiceanu', []).

ctr_arguments(
    same,
    ['VARIABLES1'-collection(var-dvar),
     'VARIABLES2'-collection(var-dvar)]).

ctr_restrictions(
    same,
    [size('VARIABLES1')=size('VARIABLES2'),
     required('VARIABLES1', var),
     required('VARIABLES2', var)]).

ctr_graph(
    same,
    ['VARIABLES1', 'VARIABLES2'],
    2,
    ['PRODUCT'>>collection(variables1, variables2)],
    [variables1^var=variables2^var],
    [for_all('CC', 'NSOURCE'='NSINK'),
     'NSOURCE'=size('VARIABLES1'),
     'NSINK'=size('VARIABLES2')]).

ctr_example(
    same,
    same(
        [[var-1], [var-9], [var-1], [var-5], [var-2], [var-1]],
        [[var-9], [var-1], [var-1], [var-1], [var-2], [var-5]])).

```

B.175 same_and_global_cardinality

```

ctr_date(same_and_global_cardinality, ['20040530']).

ctr_origin(
    same_and_global_cardinality,
    'Derived from %c and %c',
    [same, global_cardinality]).

ctr_synonyms(
    same_and_global_cardinality,
    [sgcc, same_gcc, same_and_gcc, swc, same_with_cardinalities]).

ctr_arguments(
    same_and_global_cardinality,
    ['VARIABLES1'-collection(var-dvar),
     'VARIABLES2'-collection(var-dvar),
     'VALUES'-collection(val-int, noccurrence-dvar)]).

ctr_restrictions(
    same_and_global_cardinality,
    [size('VARIABLES1')=size('VARIABLES2'),
     required('VARIABLES1', var),
     required('VARIABLES2', var),
     required('VALUES', [val, noccurrence]),
     distinct('VALUES', val),
     'VALUES' ^ noccurrence >= 0,
     'VALUES' ^ noccurrence <= size('VARIABLES1')]).

ctr_graph(
    same_and_global_cardinality,
    ['VARIABLES1', 'VARIABLES2'],
    2,
    ['PRODUCT' >> collection(variables1, variables2)],
    [variables1 ^ var = variables2 ^ var],
    [for_all('CC', 'NSOURCE' = 'NSINK'),
     'NSOURCE' = size('VARIABLES1'),
     'NSINK' = size('VARIABLES2')]).

ctr_graph(
    same_and_global_cardinality,
    ['VARIABLES1'],
    1,
    foreach('VALUES', ['SELF' >> collection(variables)]),
    [variables ^ var = 'VALUES' ^ val],
    ['NVERTEX' = 'VALUES' ^ noccurrence]).

```



```
ctr_example(  
  same_and_global_cardinality,  
  same_and_global_cardinality(  
    [[var-1], [var-9], [var-1], [var-5], [var-2], [var-1]],  
    [[var-9], [var-1], [var-1], [var-1], [var-2], [var-5]],  
    [[val-1, noccurrence-3],  
     [val-2, noccurrence-1],  
     [val-5, noccurrence-1],  
     [val-7, noccurrence-0],  
     [val-9, noccurrence-1]])).
```

B.176 same_intersection

```

ctr_date(same_intersection, ['20040530']).

ctr_origin(
  same_intersection,
  'Derived from %c and %c.',
  [same, common]).

ctr_arguments(
  same_intersection,
  ['VARIABLES1'-collection(var-dvar),
   'VARIABLES2'-collection(var-dvar)]).

ctr_restrictions(
  same_intersection,
  [required('VARIABLES1', var), required('VARIABLES2', var)]).

ctr_graph(
  same_intersection,
  ['VARIABLES1', 'VARIABLES2'],
  2,
  ['PRODUCT'>>collection(variables1, variables2)],
  [variables1^var=variables2^var],
  [for_all('CC', 'NSOURCE'='NSINK')]).

ctr_example(
  same_intersection,
  same_intersection(
    [[var-1], [var-9], [var-1], [var-5], [var-2], [var-1]],
    [[var-9],
     [var-1],
     [var-1],
     [var-1],
     [var-1],
     [var-3],
     [var-5],
     [var-8]])).

```

B.177 same_interval

```

ctr_date(same_interval, ['20030820']).

ctr_origin(same_interval, 'Derived from %c.', [same]).

ctr_arguments(
    same_interval,
    ['VARIABLES1'-collection(var-dvar),
     'VARIABLES2'-collection(var-dvar),
     'SIZE_INTERVAL'-int]).

ctr_restrictions(
    same_interval,
    [size('VARIABLES1')=size('VARIABLES2'),
     required('VARIABLES1', var),
     required('VARIABLES2', var),
     'SIZE_INTERVAL'>0]).

ctr_graph(
    same_interval,
    ['VARIABLES1', 'VARIABLES2'],
    2,
    ['PRODUCT'>>collection(variables1, variables2)],
    [(variables1^var/'SIZE_INTERVAL',
      variables2^var/'SIZE_INTERVAL')],
    [for_all('CC', 'NSOURCE'='NSINK'),
     'NSOURCE'=size('VARIABLES1'),
     'NSINK'=size('VARIABLES2')]).

ctr_example(
    same_interval,
    same_interval(
        [[var-1], [var-7], [var-6], [var-0], [var-1], [var-7]],
        [[var-8], [var-8], [var-8], [var-0], [var-1], [var-2]],
        3)).

```

B.178 same_modulo

```

ctr_date(same_modulo, ['20030820']).

ctr_origin(same_modulo, 'Derived from %c.', [same]).

ctr_arguments(
    same_modulo,
    ['VARIABLES1'-collection(var-dvar),
     'VARIABLES2'-collection(var-dvar),
     'M'-int]).

ctr_restrictions(
    same_modulo,
    [size('VARIABLES1')=size('VARIABLES2'),
     required('VARIABLES1', var),
     required('VARIABLES2', var),
     'M'>0]).

ctr_graph(
    same_modulo,
    ['VARIABLES1', 'VARIABLES2'],
    2,
    ['PRODUCT'>>collection(variables1, variables2)],
    [variables1^var mod 'M'=variables2^var mod 'M'],
    [for_all('CC', 'NSOURCE'='NSINK'),
     'NSOURCE'=size('VARIABLES1'),
     'NSINK'=size('VARIABLES2')]).

ctr_example(
    same_modulo,
    same_modulo(
        [[var-1], [var-9], [var-1], [var-5], [var-2], [var-1]],
        [[var-6], [var-4], [var-1], [var-1], [var-5], [var-5]],
        3)).

```

B.179 same_partition

```

ctr_date(same_partition, ['20030820']).

ctr_origin(same_partition, 'Derived from %c.', [same]).

ctr_types(same_partition, ['VALUES'-collection(val-int)]).

ctr_arguments(
    same_partition,
    ['VARIABLES1'-collection(var-dvar),
     'VARIABLES2'-collection(var-dvar),
     'PARTITIONS'-collection(p-'VALUES')]).

ctr_restrictions(
    same_partition,
    [required('VALUES', val),
     distinct('VALUES', val),
     size('VARIABLES1')=size('VARIABLES2'),
     required('VARIABLES1', var),
     required('VARIABLES2', var),
     required('PARTITIONS', p),
     size('PARTITIONS')>=2]).

ctr_graph(
    same_partition,
    ['VARIABLES1', 'VARIABLES2'],
    2,
    ['PRODUCT'>>collection(variables1, variables2)],
    [in_same_partition(
        variables1^var,
        variables2^var,
        'PARTITIONS')],
    [for_all('CC', 'NSOURCE'='NSINK'),
     'NSOURCE'=size('VARIABLES1'),
     'NSINK'=size('VARIABLES2')]).

ctr_example(
    same_partition,
    same(
        [[var-1], [var-2], [var-6], [var-3], [var-1], [var-2]],
        [[var-6], [var-6], [var-2], [var-3], [var-1], [var-3]],
        [p-[[val-1], [val-3]]],
        [p-[[val-4]]],
        [p-[[val-2], [val-6]]]]).

```

B.180 sequence_folding

```

ctr_automaton(sequence_folding, sequence_folding) .

ctr_date(sequence_folding, ['20030820', '20040530']) .

ctr_origin(sequence_folding, 'J.~Pearson', []).

ctr_arguments(
    sequence_folding,
    ['LETTERS'-collection(index-int, next-dvar)]).

ctr_restrictions(
    sequence_folding,
    [size('LETTERS')>=1,
     required('LETTERS', [index, next]),
     'LETTERS'^index>=1,
     'LETTERS'^index<=size('LETTERS'),
     increasing_seq('LETTERS', index),
     'LETTERS'^next>=1,
     'LETTERS'^next<=size('LETTERS')]).

ctr_graph(
    sequence_folding,
    ['LETTERS'],
    1,
    ['SELF'>>collection(letters)],
    [letters^next>=letters^index],
    ['NARC'=size('LETTERS')]).

ctr_graph(
    sequence_folding,
    ['LETTERS'],
    2,
    ['CLIQUE'(<)>>collection(letters1, letters2)],
    [#\/(letters2^index>=letters1^next,
        letters2^next<=letters1^next)],
    ['NARC'=size('LETTERS')*(size('LETTERS')-1)/2]).

ctr_example(
    sequence_folding,
    sequence_folding(
        [[index-1, next-1],
         [index-2, next-8],
         [index-3, next-3],
         [index-4, next-5],

```

```
[index-5,next-5],
[index-6,next-7],
[index-7,next-7],
[index-8,next-8],
[index-9,next-9]]).
```

```
sequence_folding(A) :-
  sequence_folding_signature(A,B),
  automaton(
    B,
    C,
    B,
    0..2,
    [source(s),sink(t)],
    [arc(s,0,s),arc(s,1,s),arc(s,$,t)],
    [],
    [],
    []).
```

```
sequence_folding_signature([],[]).
```

```
sequence_folding_signature([A],[]).
```

```
sequence_folding_signature([A,B|C],D) :-
  sequence_folding_signature([B|C],A,E),
  sequence_folding_signature([B|C],F),
  append(E,F,D).
```

```
sequence_folding_signature([],A,[]).
```

```
sequence_folding_signature([A|B],C,[D|E]) :-
  C=[index-F,next-G],
  A=[index-H,next-I],
  F#=<G#\H#=<I#\G#=<H#<=>D#=0,
  F#=<G#\H#=<I#\G#>H#\I#=<G#<=>D#=1,
  sequence_folding_signature(B,C,E).
```

B.181 set_value_precede

```
ctr_predefined(set_value_precede).  
  
ctr_date(set_value_precede, ['20041003']).  
  
ctr_origin(set_value_precede, '\\cite{YatChiuLawJimmyLee04}', []).  
  
ctr_arguments(  
    set_value_precede,  
    ['S'-int, 'T'-int, 'VARIABLES'-collection(var-svar)]).  
  
ctr_restrictions(  
    set_value_precede,  
    ['S'=\='T', required('VARIABLES', var)]).  
  
ctr_example(  
    set_value_precede,  
    set_value_precede(  
        2,  
        1,  
        [[var-{0,2}], [var-{0,1}], [var-{}], [var-{1}]])).
```


B.182 shift

```

ctr_date(shift, ['20030820']).

ctr_origin(shift, 'N.~Beldiceanu', []).

ctr_arguments(
    shift,
    ['MIN_BREAK'-int,
     'MAX_RANGE'-int,
     'TASKS'-collection(id-int,origin-dvar,end-dvar)]).

ctr_restrictions(
    shift,
    ['MIN_BREAK'>0,
     'MAX_RANGE'>0,
     required('TASKS', [id,origin,end]),
     distinct('TASKS',id)].

ctr_graph(
    shift,
    ['TASKS'],
    1,
    ['SELF'>>collection(tasks)],
    [tasks^end>=tasks^origin,
     tasks^end-tasks^origin=<'MAX_RANGE'],
    ['NARC'=size('TASKS')]).

ctr_graph(
    shift,
    ['TASKS'],
    2,
    ['CLIQUE'>>collection(tasks1,tasks2)],
    [#\/(#\/(#\/(tasks2^origin>=tasks1^end,
                tasks2^origin-tasks1^end=<'MIN_BREAK'),
                #\/(tasks1^origin>=tasks2^end,
                tasks1^origin-tasks2^end=<'MIN_BREAK'))),
     tasks2^origin<tasks1^end#\/tasks1^origin<tasks2^end)],
    [],
    [>>('CC',
         [-(variables,
            col('VARIABLES'-collection(var-dvar),
              [item(var-'TASKS'^origin),
               item(var-'TASKS'^end)]))]),
     [range_ctr(variables,=<,'MAX_RANGE')]).

```

```
ctr_example(  
  shift,  
  shift(  
    6,  
    8,  
    [[id-1,origin-17,end-20],  
     [id-2,origin-7,end-10],  
     [id-3,origin-2,end-4],  
     [id-4,origin-21,end-22],  
     [id-5,origin-5,end-6]])).
```

B.183 size_maximal_sequence_alldifferent

```
ctr_date(size_maximal_sequence_alldifferent, ['20030820']).

ctr_origin(
    size_maximal_sequence_alldifferent,
    'N.~Beldiceanu',
    []).

ctr_synonyms(
    size_maximal_sequence_alldifferent,
    [size_maximal_sequence_alldiff,
     size_maximal_sequence_alldistinct]).

ctr_arguments(
    size_maximal_sequence_alldifferent,
    ['SIZE'-dvar, 'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    size_maximal_sequence_alldifferent,
    ['SIZE'>=0,
     'SIZE'<=size('VARIABLES'),
     required('VARIABLES', var)]).

ctr_graph(
    size_maximal_sequence_alldifferent,
    ['VARIABLES'],
    *,
    ['PATH_N'>>collection],
    [alldifferent(collection)],
    ['NARC'='SIZE']).

ctr_example(
    size_maximal_sequence_alldifferent,
    size_maximal_sequence_alldifferent(
        4,
        [[var-2],
         [var-2],
         [var-4],
         [var-5],
         [var-2],
         [var-7],
         [var-4]])).
```

B.184 size_maximal_starting_sequence_alldifferent

```

ctr_date(
  size_maximal_starting_sequence_alldifferent,
  ['20030820']).

ctr_origin(
  size_maximal_starting_sequence_alldifferent,
  'N.~Beldiceanu',
  []).

ctr_synonyms(
  size_maximal_starting_sequence_alldifferent,
  [size_maximal_starting_sequence_alldiff,
   size_maximal_starting_sequence_alldistinct]).

ctr_arguments(
  size_maximal_starting_sequence_alldifferent,
  ['SIZE'-dvar,'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
  size_maximal_starting_sequence_alldifferent,
  ['SIZE'>=0,
   'SIZE'=<size('VARIABLES'),
   required('VARIABLES',var)]).

ctr_graph(
  size_maximal_starting_sequence_alldifferent,
  ['VARIABLES'],
  *,
  ['PATH_1'>>collection],
  [alldifferent(collection)],
  ['NARC'='SIZE']).

ctr_example(
  size_maximal_starting_sequence_alldifferent,
  size_maximal_starting_sequence_alldifferent(
    4,
    [[var-9],
     [var-2],
     [var-4],
     [var-5],
     [var-2],
     [var-7],
     [var-4]])).

```

B.185 sliding_card_skip0

```

ctr_automaton(sliding_card_skip0,sliding_card_skip0).

ctr_date(sliding_card_skip0,['20000128','20030820','20040530']).

ctr_origin(sliding_card_skip0,'N.~Beldiceanu',[]).

ctr_arguments(
    sliding_card_skip0,
    ['ATLEAST'-int,
     'ATMOST'-int,
     'VARIABLES'-collection(var-dvar),
     'VALUES'-collection(val-int)].

ctr_restrictions(
    sliding_card_skip0,
    ['ATLEAST'>=0,
     'ATMOST'>='ATLEAST',
     required('VARIABLES',var),
     required('VALUES',val),
     distinct('VALUES',val),
     'VALUES'^val=\=0]).

ctr_graph(
    sliding_card_skip0,
    ['VARIABLES'],
    2,
    ['PATH'>>collection(variables1,variables2),
     'LOOP'>>collection(variables1,variables2)],
    [variables1^var=\=0,variables2^var=\=0],
    [],
    ['CC'>>[variables]],
    [among_low_up('ATLEAST','ATMOST',variables,'VALUES')]).

ctr_example(
    sliding_card_skip0,
    sliding_card_skip0(
        2,
        3,
        [[var-0],
         [var-7],
         [var-2],
         [var-9],
         [var-0],
         [var-0],

```

```

    [var-9],
    [var-4],
    [var-9]],
    [[val-7],[val-9]]) .

```

```

sliding_card_skip0(A,B,C,D) :-
    col_to_list(D,E),
    list_to_fdset(E,F),
    sliding_card_skip0_signature(C,G,F),
    automaton(
        G,
        H,
        G,
        0..2,
        [source(s),node(i),sink(t)],
        [arc(s,0,s),
         arc(s,1,i,[0]),
         arc(s,2,i,[1]),
         arc(s,$,t),
         arc(i,0,s,(in(I,A..B)->[I])),
         arc(i,1,i),
         arc(i,2,i,[I+1]),
         arc(i,$,t,(in(I,A..B)->[I]))],
        [I],
        [0],
        [J]) .

```

```

sliding_card_skip0_signature([],[],A) .

```

```

sliding_card_skip0_signature([[var-A]|B],[C|D],E) :-
    A#\=0#\<=>F,
    in_set(A,E)#\<=>G,
    in(C,0..2),
    C#\=max(2*F+G-1,0),
    sliding_card_skip0_signature(B,D,E) .

```

B.186 sliding_distribution

```

ctr_date(sliding_distribution, ['20031008']).

ctr_origin(sliding_distribution, '\\cite{ReginPuget97}', []).

ctr_arguments(
  sliding_distribution,
  ['SEQ'-int,
   'VARIABLES'-collection(var-dvar),
   'VALUES'-collection(val-int, omin-int, omax-int)].

ctr_restrictions(
  sliding_distribution,
  ['SEQ'>0,
   'SEQ'=<size('VARIABLES'),
   required('VARIABLES', var),
   size('VALUES')>0,
   required('VALUES', [val, omin, omax]),
   distinct('VALUES', val),
   'VALUES' ^omin>=0,
   'VALUES' ^omax=<'SEQ',
   'VALUES' ^omin=<'VALUES' ^omax]).

ctr_graph(
  sliding_distribution,
  ['VARIABLES'],
  'SEQ',
  ['PATH'>>collection],
  [global_cardinality_low_up(collection, 'VALUES')],
  ['NARC'=size('VARIABLES')-'SEQ'+1]).

ctr_example(
  sliding_distribution,
  sliding_distribution(
    4,
    [[var-0],
     [var-5],
     [var-6],
     [var-6],
     [var-5],
     [var-0],
     [var-0]],
    [[val-0, omin-1, omax-2],
     [val-1, omin-0, omax-4],
     [val-4, omin-0, omax-4],

```

```
[val-5,omin-1,omax-2],  
[val-6,omin-0,omax-2]]) .
```


B.187 sliding_sum

```

ctr_date(sliding_sum, ['20000128', '20030820']).

ctr_origin(sliding_sum, 'CHIP', []).

ctr_arguments(
    sliding_sum,
    ['LOW'-int,
     'UP'-int,
     'SEQ'-int,
     'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    sliding_sum,
    ['UP'>='LOW',
     'SEQ'>0,
     'SEQ'=<size('VARIABLES'),
     required('VARIABLES', var)]).

ctr_graph(
    sliding_sum,
    ['VARIABLES'],
    'SEQ',
    ['PATH'>>collection],
    [sum_ctr(collection, >=, 'LOW'), sum_ctr(collection, <=, 'UP')],
    ['NARC'=size('VARIABLES')-'SEQ'+1]).

ctr_example(
    sliding_sum,
    sliding_sum(
        3,
        7,
        4,
        [[var-1],
         [var-4],
         [var-2],
         [var-0],
         [var-0],
         [var-3],
         [var-4]])).

```

B.188 sliding_time_window

```

ctr_date(sliding_time_window, ['20030820']).

ctr_origin(sliding_time_window, 'N.~Beldiceanu', []).

ctr_arguments(
    sliding_time_window,
    ['WINDOW_SIZE'-int,
     'LIMIT'-int,
     'TASKS'-collection(id-int,origin-dvar,duration-dvar)]).

ctr_restrictions(
    sliding_time_window,
    ['WINDOW_SIZE'>0,
     'LIMIT'>=0,
     required('TASKS',[id,origin,duration]),
     distinct('TASKS',id),
     'TASKS'^duration>=0]).

ctr_graph(
    sliding_time_window,
    ['TASKS'],
    2,
    ['CLIQUE'>>collection(tasks1,tasks2)],
    [tasks1^origin=<tasks2^origin,
     tasks2^origin-tasks1^origin<'WINDOW_SIZE'],
    [],
    ['SUCC'>>[source,tasks]],
    [sliding_time_window_from_start(
     'WINDOW_SIZE',
     'LIMIT',
     tasks,
     source^origin)]).

ctr_example(
    sliding_time_window,
    sliding_time_window(
        9,
        6,
        [[id-1,origin-10,duration-3],
         [id-2,origin-5,duration-1],
         [id-3,origin-6,duration-2],
         [id-4,origin-14,duration-2],
         [id-5,origin-2,duration-2]])).

```

B.189 sliding_time_window_from_start

```

ctr_date(sliding_time_window_from_start, ['20030820']).

ctr_origin(
    sliding_time_window_from_start,
    'Used for defining %c.',
    [sliding_time_window]).

ctr_arguments(
    sliding_time_window_from_start,
    ['WINDOW_SIZE'-int,
     'LIMIT'-int,
     'TASKS'-collection(id-int,origin-dvar,duration-dvar),
     'START'-dvar]).

ctr_restrictions(
    sliding_time_window_from_start,
    ['WINDOW_SIZE'>0,
     'LIMIT'>=0,
     required('TASKS',[id,origin,duration]),
     distinct('TASKS',id),
     'TASKS'^duration>=0]).

ctr_derived_collections(
    sliding_time_window_from_start,
    [col('S'-collection(var-dvar),[item(var-'START')])]).

ctr_graph(
    sliding_time_window_from_start,
    ['S','TASKS'],
    2,
    ['PRODUCT'>>collection(s,tasks)],
    ['TRUE'],
    [=<('SUM_WEIGHT_ARC'(
        max(0,
            -(min(s^var+'WINDOW_SIZE',
                  tasks^origin+tasks^duration),
                max(s^var,tasks^origin))))),
     'LIMIT']]).

ctr_example(
    sliding_time_window_from_start,
    sliding_time_window(
        9,
        6,

```

```
[[id-1,origin-10,duration-3],  
 [id-2,origin-5,duration-1],  
 [id-3,origin-6,duration-2]],  
5)).
```

B.190 sliding_time_window_sum

```

ctr_date(sliding_time_window_sum, ['20030820']).

ctr_origin(
    sliding_time_window_sum,
    'Derived from %c.',
    [sliding_time_window]).

ctr_arguments(
    sliding_time_window_sum,
    ['WINDOW_SIZE'-int,
     'LIMIT'-int,
     -('TASKS',
      collection(id-int,origin-dvar,end-dvar,npoint-dvar))]).

ctr_restrictions(
    sliding_time_window_sum,
    ['WINDOW_SIZE'>0,
     'LIMIT'>=0,
     required('TASKS', [id,origin,end,npoint]),
     distinct('TASKS',id),
     'TASKS'^npoint>=0]).

ctr_graph(
    sliding_time_window_sum,
    ['TASKS'],
    1,
    ['SELF'>>collection(tasks)],
    [tasks^origin=<tasks^end],
    ['NARC'=size('TASKS')]).

ctr_graph(
    sliding_time_window_sum,
    ['TASKS'],
    2,
    ['CLIQUE'>>collection(tasks1,tasks2)],
    [tasks1^end=<tasks2^end,
     tasks2^origin-tasks1^end<'WINDOW_SIZE'-1],
    [],
    [>>('SUCC',
        [source,
         -(variables,
            col('VARIABLES'-collection(var-dvar),
                [item(var-'TASKS'^npoint)])])]),
     [sum_ctr(variables,=<,'LIMIT')]).

```

```
ctr_example(  
  sliding_time_window_sum,  
  sliding_time_window_sum(  
    9,  
    16,  
    [[id-1,origin-10,end-13,npoint-2],  
     [id-2,origin-5,end-6,npoint-3],  
     [id-3,origin-6,end-8,npoint-4],  
     [id-4,origin-14,end-16,npoint-5],  
     [id-5,origin-2,end-4,npoint-6]])).
```

B.191 smooth

```

ctr_automaton(smooth, smooth) .

ctr_date(smooth, ['20000128', '20030820', '20040530']) .

ctr_origin(smooth, 'Derived from %c.', [change]) .

ctr_arguments(
    smooth,
    ['NCHANGE' -dvar,
     'TOLERANCE' -int,
     'VARIABLES' -collection(var-dvar)] .

ctr_restrictions(
    smooth,
    ['NCHANGE' >=0,
     'NCHANGE' <size('VARIABLES'),
     'TOLERANCE' >=0,
     required('VARIABLES', var)] .

ctr_graph(
    smooth,
    ['VARIABLES'],
    2,
    ['PATH' >>collection(variables1, variables2)],
    [abs(variables1^var-variables2^var) > 'TOLERANCE'],
    ['NARC' = 'NCHANGE']) .

ctr_example(
    smooth,
    smooth(1, 2, [[var-1], [var-3], [var-4], [var-5], [var-2]])) .

smooth(A, B, C) :-
    smooth_signature(C, D, B),
    automaton(
        D,
        E,
        D,
        0..1,
        [source(s), sink(t)],
        [arc(s, 1, s, [F+1]), arc(s, 0, s), arc(s, $, t)],
        [F],
        [0],
        [A]) .

```

```
smooth_signature([], [], A).
```

```
smooth_signature([A], [], B).
```

```
smooth_signature([[var-A], [var-B] | C], [D | E], F) :-  
    abs(A-B) #>F #<=>D#=1,  
    smooth_signature([[var-B] | C], E, F).
```


B.192 soft_alldifferent_ctr

```

ctr_date(soft_alldifferent_ctr, ['20030820']).

ctr_origin(
    soft_alldifferent_ctr,
    '\\cite{PetitReginBessiere01}',
    []).

ctr_synonyms(
    soft_alldifferent_ctr,
    [soft_alldiff_ctr, soft_alldistinct_ctr]).

ctr_arguments(
    soft_alldifferent_ctr,
    ['C'-dvar, 'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    soft_alldifferent_ctr,
    ['C'>=0,
     =<('C',
        /(- (size('VARIABLES') * size('VARIABLES')),
            size('VARIABLES'))),
        2)),
    required('VARIABLES', var)].

ctr_graph(
    soft_alldifferent_ctr,
    ['VARIABLES'],
    2,
    ['CLIQUE' (<)>>collection(variables1, variables2)],
    [variables1^var=variables2^var],
    ['NARC'='C']).

ctr_example(
    soft_alldifferent_ctr,
    soft_alldifferent_ctr(
        4,
        [[var-5], [var-1], [var-9], [var-1], [var-5], [var-5]])).

```

B.193 soft_alldifferent_var

```

ctr_date(soft_alldifferent_var, ['20030820']).

ctr_origin(
    soft_alldifferent_var,
    '\\cite{PetitReginBessiere01}',
    []).

ctr_synonyms(
    soft_alldifferent_var,
    [soft_alldiff_var, soft_alldistinct_var]).

ctr_arguments(
    soft_alldifferent_var,
    ['C'-dvar, 'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    soft_alldifferent_var,
    ['C'>=0, 'C'<size('VARIABLES'), required('VARIABLES', var)]).

ctr_graph(
    soft_alldifferent_var,
    ['VARIABLES'],
    2,
    ['CLIQUE'>>collection(variables1, variables2)],
    [variables1^var=variables2^var],
    ['NSCC'=size('VARIABLES')-'C']).

ctr_example(
    soft_alldifferent_var,
    soft_alldifferent_var(
        3,
        [[var-5], [var-1], [var-9], [var-1], [var-5], [var-5]])).

```

B.194 soft_same_interval_var

```

ctr_date(soft_same_interval_var, ['20050507']).

ctr_origin(
    soft_same_interval_var,
    'Derived from %c',
    [same_interval]).

ctr_synonyms(soft_same_interval_var, [soft_same_interval]).

ctr_arguments(
    soft_same_interval_var,
    ['C'-dvar,
     'VARIABLES1'-collection(var-dvar),
     'VARIABLES2'-collection(var-dvar),
     'SIZE_INTERVAL'-int]).

ctr_restrictions(
    soft_same_interval_var,
    ['C'>=0,
     'C'=<size('VARIABLES1'),
     size('VARIABLES1')=size('VARIABLES2'),
     required('VARIABLES1', var),
     required('VARIABLES2', var),
     'SIZE_INTERVAL'>0]).

ctr_graph(
    soft_same_interval_var,
    ['VARIABLES1', 'VARIABLES2'],
    2,
    ['PRODUCT'>>collection(variables1, variables2)],
    [(variables1^var/'SIZE_INTERVAL',
      variables2^var/'SIZE_INTERVAL')],
    ['NSINK_NSOURCE'=size('VARIABLES1')-'C']).

ctr_example(
    soft_same_interval_var,
    soft_same_interval_var(
        4,
        [[var-9], [var-9], [var-9], [var-9], [var-9], [var-1]],
        [[var-9], [var-1], [var-1], [var-1], [var-1], [var-8]],
        3)).

```

B.195 soft_same_modulo_var

```

ctr_date(soft_same_modulo_var, ['20050507']).

ctr_origin(
    soft_same_modulo_var,
    'Derived from %c',
    [same_modulo]).

ctr_synonyms(soft_same_modulo_var, [soft_same_modulo]).

ctr_arguments(
    soft_same_modulo_var,
    ['C'-dvar,
     'VARIABLES1'-collection(var-dvar),
     'VARIABLES2'-collection(var-dvar),
     'M'-int]).

ctr_restrictions(
    soft_same_modulo_var,
    ['C'>=0,
     'C'<=size('VARIABLES1'),
     size('VARIABLES1')=size('VARIABLES2'),
     required('VARIABLES1', var),
     required('VARIABLES2', var),
     'M'>0]).

ctr_graph(
    soft_same_modulo_var,
    ['VARIABLES1', 'VARIABLES2'],
    2,
    ['PRODUCT'>>collection(variables1, variables2)],
    [variables1^var mod 'M'=variables2^var mod 'M'],
    ['NSINK_NSOURCE'=size('VARIABLES1')-'C']).

ctr_example(
    soft_same_modulo_var,
    soft_same_modulo_var(
        4,
        [[var-9], [var-9], [var-9], [var-9], [var-9], [var-1]],
        [[var-9], [var-1], [var-1], [var-1], [var-1], [var-8]],
        3)).

```

B.196 soft_same_partition_var

```

ctr_date(soft_same_partition_var, ['20050507']).

ctr_origin(
    soft_same_partition_var,
    'Derived from %c',
    [same_partition]).

ctr_synonyms(soft_same_partition_var, [soft_same_partition]).

ctr_types(
    soft_same_partition_var,
    ['VALUES'-collection(val-int)]).

ctr_arguments(
    soft_same_partition_var,
    ['C'-dvar,
     'VARIABLES1'-collection(var-dvar),
     'VARIABLES2'-collection(var-dvar),
     'PARTITIONS'-collection(p-'VALUES')]).

ctr_restrictions(
    soft_same_partition_var,
    ['C'>=0,
     'C'<size('VARIABLES1'),
     size('VARIABLES1')=size('VARIABLES2'),
     required('VARIABLES1', var),
     required('VARIABLES2', var),
     required('PARTITIONS', p),
     size('PARTITIONS')>=2,
     required('VALUES', val),
     distinct('VALUES', val)]).

ctr_graph(
    soft_same_partition_var,
    ['VARIABLES1', 'VARIABLES2'],
    2,
    ['PRODUCT'>>collection(variables1, variables2)],
    [in_same_partition(
        variables1^var,
        variables2^var,
        'PARTITIONS')],
    ['NSINK_NSOURCE'=size('VARIABLES1')-'C'])).

ctr_example(

```

```
soft_same_partition_var,  
soft_same_partition_var(  
  4,  
  [[var-9],[var-9],[var-9],[var-9],[var-9],[var-1]],  
  [[var-9],[var-1],[var-1],[var-1],[var-1],[var-8]],  
  [[p-[[val-1],[val-2]]],  
   [p-[[val-9]]],  
   [p-[[val-7],[val-8]]]])).
```

B.197 soft_same_var

```

ctr_date(soft_same_var, ['20050507']).

ctr_origin(soft_same_var, '\\cite{vanHoeve05}', []).

ctr_synonyms(soft_same_var, [soft_same]).

ctr_arguments(
    soft_same_var,
    ['C'-dvar,
     'VARIABLES1'-collection(var-dvar),
     'VARIABLES2'-collection(var-dvar)]).

ctr_restrictions(
    soft_same_var,
    ['C'>=0,
     'C'=<size('VARIABLES1'),
     size('VARIABLES1')=size('VARIABLES2'),
     required('VARIABLES1', var),
     required('VARIABLES2', var)]).

ctr_graph(
    soft_same_var,
    ['VARIABLES1', 'VARIABLES2'],
    2,
    ['PRODUCT'>>collection(variables1, variables2)],
    [variables1^var=variables2^var],
    ['NSINK_NSOURCE'=size('VARIABLES1')-'C']).

ctr_example(
    soft_same_var,
    soft_same_var(
        4,
        [[var-9], [var-9], [var-9], [var-9], [var-9], [var-1]],
        [[var-9], [var-1], [var-1], [var-1], [var-1], [var-8]]).

```

B.198 soft_used_by_interval_var

```
ctr_date(soft_used_by_interval_var, ['20050507']).
```

```
ctr_origin(
    soft_used_by_interval_var,
    'Derived from %c.',
    [used_by_interval]).
```

```
ctr_synonyms(soft_used_by_interval_var, [soft_used_by_interval]).
```

```
ctr_arguments(
    soft_used_by_interval_var,
    ['C'-dvar,
     'VARIABLES1'-collection(var-dvar),
     'VARIABLES2'-collection(var-dvar),
     'SIZE_INTERVAL'-int]).
```

```
ctr_restrictions(
    soft_used_by_interval_var,
    ['C'>=0,
     'C'<=size('VARIABLES2'),
     size('VARIABLES1')>=size('VARIABLES2'),
     required('VARIABLES1', var),
     required('VARIABLES2', var),
     'SIZE_INTERVAL'>0]).
```

```
ctr_graph(
    soft_used_by_interval_var,
    ['VARIABLES1', 'VARIABLES2'],
    2,
    ['PRODUCT'>>collection(variables1, variables2)],
    [(variables1^var/'SIZE_INTERVAL',
      variables2^var/'SIZE_INTERVAL')],
    ['NSINK_NSOURCE'=size('VARIABLES2')-'C']).
```

```
ctr_example(
    soft_used_by_interval_var,
    soft_used_by_interval_var(
        2,
        [[var-9], [var-1], [var-1], [var-8], [var-8]],
        [[var-9], [var-9], [var-9], [var-1]],
        3)).
```


B.199 soft_used_by_modulo_var

```

ctr_date(soft_used_by_modulo_var, ['20050507']).

ctr_origin(
    soft_used_by_modulo_var,
    'Derived from %c',
    [used_by_modulo]).

ctr_synonyms(soft_used_by_modulo_var, [soft_used_by_modulo]).

ctr_arguments(
    soft_used_by_modulo_var,
    ['C'-dvar,
     'VARIABLES1'-collection(var-dvar),
     'VARIABLES2'-collection(var-dvar),
     'M'-int]).

ctr_restrictions(
    soft_used_by_modulo_var,
    ['C'>=0,
     'C'=<size('VARIABLES2'),
     size('VARIABLES1')>=size('VARIABLES2'),
     required('VARIABLES1', var),
     required('VARIABLES2', var),
     'M'>0]).

ctr_graph(
    soft_used_by_modulo_var,
    ['VARIABLES1', 'VARIABLES2'],
    2,
    ['PRODUCT'>>collection(variables1, variables2)],
    [variables1^var mod 'M'=variables2^var mod 'M'],
    ['NSINK_NSOURCE'=size('VARIABLES2')-'C']).

ctr_example(
    soft_used_by_modulo_var,
    soft_used_by_modulo_var(
        2,
        [[var-9], [var-1], [var-1], [var-8], [var-8]],
        [[var-9], [var-9], [var-9], [var-1]],
        3)).

```

B.200 `soft_used_by_partition_var`

```

ctr_date(soft_used_by_partition_var, ['20050507']).

ctr_origin(
    soft_used_by_partition_var,
    'Derived from %c.',
    [used_by_partition]).

ctr_synonyms(
    soft_used_by_partition_var,
    [soft_used_by_partition]).

ctr_types(
    soft_used_by_partition_var,
    ['VALUES'-collection(val-int)]).

ctr_arguments(
    soft_used_by_partition_var,
    ['C'-dvar,
     'VARIABLES1'-collection(var-dvar),
     'VARIABLES2'-collection(var-dvar),
     'PARTITIONS'-collection(p-'VALUES')]).

ctr_restrictions(
    soft_used_by_partition_var,
    ['C'>=0,
     'C'<=size('VARIABLES2'),
     size('VARIABLES1')>=size('VARIABLES2'),
     required('VARIABLES1', var),
     required('VARIABLES2', var),
     required('PARTITIONS', p),
     size('PARTITIONS')>=2,
     required('VALUES', val),
     distinct('VALUES', val)]).

ctr_graph(
    soft_used_by_partition_var,
    ['VARIABLES1', 'VARIABLES2'],
    2,
    ['PRODUCT'>>collection(variables1, variables2)],
    [in_same_partition(
        variables1^var,
        variables2^var,
        'PARTITIONS')],
    ['NSINK_NSOURCE'=size('VARIABLES2')-'C']).

```

```
ctr_example(  
  soft_used_by_partition_var,  
  soft_used_by_partition_var(  
    2,  
    [[var-9], [var-1], [var-1], [var-8], [var-8]],  
    [[var-9], [var-9], [var-9], [var-1]],  
    [[p-[[val-1], [val-2]]],  
     [p-[[val-9]]],  
     [p-[[val-7], [val-8]]]])).  
)
```

B.201 soft_used_by_var

```

ctr_date(soft_used_by_var, ['20050507']).

ctr_origin(soft_used_by_var, 'Derived from %c', [used_by]).

ctr_synonyms(soft_used_by_var, [soft_used_by]).

ctr_arguments(
    soft_used_by_var,
    ['C'-dvar,
     'VARIABLES1'-collection(var-dvar),
     'VARIABLES2'-collection(var-dvar)]).

ctr_restrictions(
    soft_used_by_var,
    ['C'>=0,
     'C'=<size('VARIABLES2'),
     size('VARIABLES1')>=size('VARIABLES2'),
     required('VARIABLES1', var),
     required('VARIABLES2', var)]).

ctr_graph(
    soft_used_by_var,
    ['VARIABLES1', 'VARIABLES2'],
    2,
    ['PRODUCT'>>collection(variables1, variables2)],
    [variables1^var=variables2^var],
    ['NSINK_NSOURCE'=size('VARIABLES2')-'C']).

ctr_example(
    soft_used_by_var,
    soft_used_by_var(
        2,
        [[var-9], [var-1], [var-1], [var-8], [var-8]],
        [[var-9], [var-9], [var-9], [var-1]])).

```

B.202 sort

```

ctr_date(sort, ['20030820']).

ctr_origin(sort, '\\cite{OlderSwinkelsEmden95}', []).

ctr_arguments(
    sort,
    ['VARIABLES1'-collection(var-dvar),
     'VARIABLES2'-collection(var-dvar)]).

ctr_restrictions(
    sort,
    [size('VARIABLES1')=size('VARIABLES2'),
     required('VARIABLES1', var),
     required('VARIABLES2', var)]).

ctr_graph(
    sort,
    ['VARIABLES1', 'VARIABLES2'],
    2,
    ['PRODUCT' >> collection(variables1, variables2)],
    [variables1^var=variables2^var],
    [for_all('CC', 'NSOURCE'='NSINK'),
     'NSOURCE'=size('VARIABLES1'),
     'NSINK'=size('VARIABLES2')]).

ctr_graph(
    sort,
    ['VARIABLES2'],
    2,
    ['PATH' >> collection(variables1, variables2)],
    [variables1^var=<variables2^var],
    ['NARC'=size('VARIABLES2')-1]).

ctr_example(
    sort,
    sort(
        [[var-1], [var-9], [var-1], [var-5], [var-2], [var-1]],
        [[var-1], [var-1], [var-1], [var-2], [var-5], [var-9]])).

```

B.203 `sort_permutation`

```

ctr_date(sort_permutation, ['20030820']).

ctr_origin(sort_permutation, '\\cite{Zhou97}', []).

ctr_usual_name(sort_permutation, sort).

ctr_arguments(
    sort_permutation,
    ['FROM'-collection(var-dvar),
     'PERMUTATION'-collection(var-dvar),
     'TO'-collection(var-dvar)]).

ctr_restrictions(
    sort_permutation,
    [size('PERMUTATION')=size('FROM'),
     size('PERMUTATION')=size('TO'),
     'PERMUTATION'^var>=1,
     'PERMUTATION'^var<size('PERMUTATION'),
     alldifferent('PERMUTATION'),
     required('FROM', var),
     required('PERMUTATION', var),
     required('TO', var)]).

ctr_derived_collections(
    sort_permutation,
    [col('FROM_PERMUTATION'-collection(var-dvar, ind-dvar),
        [item(var-'FROM'^var, ind-'PERMUTATION'^var)])]).

ctr_graph(
    sort_permutation,
    ['FROM_PERMUTATION', 'TO'],
    2,
    ['PRODUCT'>>collection(from_permutation, to)],
    [from_permutation^var=to^var, from_permutation^ind=to^key],
    ['NARC'=size('PERMUTATION')]).

ctr_graph(
    sort_permutation,
    ['TO'],
    2,
    ['PATH'>>collection(to1, to2)],
    [to1^var<to2^var],
    ['NARC'=size('TO')-1]).

```

```
ctr_example(  
  sort_permutation,  
  sort_permutation(  
    [[var-1], [var-9], [var-1], [var-5], [var-2], [var-1]],  
    [[var-1], [var-6], [var-3], [var-5], [var-4], [var-2]],  
    [[var-1], [var-1], [var-1], [var-2], [var-5], [var-9]])).
```

B.204 stage_element

```

ctr_automaton(stage_element, stage_element).

ctr_date(stage_element, ['20040828']).

ctr_origin(stage_element, 'CHOCO, derived from %c.', [element]).

ctr_usual_name(stage_element, stage_elt).

ctr_arguments(
    stage_element,
    ['ITEM'-collection(index-dvar, value-dvar),
     'TABLE'-collection(low-int, up-int, value-int)]).

ctr_restrictions(
    stage_element,
    [required('ITEM', [index, value]),
     size('ITEM')=1,
     required('TABLE', [low, up, value])]).

ctr_graph(
    stage_element,
    ['TABLE'],
    2,
    ['PATH'>>collection(table1, table2)],
    [table1^low=<table1^up,
     table1^up+1=table2^low,
     table2^low=<table2^up],
    ['NARC'=size('TABLE')-1]).

ctr_graph(
    stage_element,
    ['ITEM', 'TABLE'],
    2,
    ['PRODUCT'>>collection(item, table)],
    [item^index>=table^low,
     item^index=<table^up,
     item^value=table^value],
    ['NARC'=1]).

ctr_example(
    stage_element,
    stage_element(
        [[index-5, value-6]],
        [[low-3, up-7, value-6]],

```



```
[low-8, up-8, value-9],
[low-9, up-14, value-2],
[low-15, up-19, value-9]]).
```

```
stage_element(A,B) :-
  A=[[index-C,value-D]],
  stage_element_signature(B,E,C,D),
  automaton(
    E,
    F,
    E,
    0..1,
    [source(s),sink(t)],
    [arc(s,0,s),arc(s,1,t)],
    [],
    [],
    []).
```

```
stage_element_signature([],[],A,B).
```

```
stage_element_signature([[low-A,up-B,value-C]|D],[E|F],G,H) :-
  A#=<G#/\G#=<B#/\H#=C#<=>E,
  stage_element_signature(D,F,G,H).
```

B.205 stretch_circuit

```

ctr_date(stretch_circuit, ['20030820']).

ctr_origin(stretch_circuit, '\\cite{Pesant01}', []).

ctr_usual_name(stretch_circuit, stretch).

ctr_arguments(
    stretch_circuit,
    ['VARIABLES'-collection(var-dvar),
     'VALUES'-collection(val-int, lmin-int, lmax-int)]).

ctr_restrictions(
    stretch_circuit,
    [size('VARIABLES')>0,
     required('VARIABLES', var),
     size('VALUES')>0,
     required('VALUES', [val, lmin, lmax]),
     distinct('VALUES', val),
     'VALUES'^lmin=<'VALUES'^lmax]).

ctr_graph(
    stretch_circuit,
    ['VARIABLES'],
    2,
    foreach(
        'VALUES',
        ['CIRCUIT'>>collection(variables1, variables2),
         'LOOP'>>collection(variables1, variables2)]),
    [variables1^var='VALUES'^val, variables2^var='VALUES'^val],
    [not_in('MIN_NCC', 1, 'VALUES'^lmin-1),
     'MAX_NCC'=<'VALUES'^lmax]).

ctr_example(
    stretch_circuit,
    stretch_circuit(
        [[var-6],
         [var-6],
         [var-3],
         [var-1],
         [var-1],
         [var-1],
         [var-6],
         [var-6]],
        [[val-1, lmin-2, lmax-4],

```

```
[val-2, lmin-2, lmax-3],  
[val-3, lmin-1, lmax-6],  
[val-6, lmin-2, lmax-4]]) .
```

B.206 stretch_path

```

ctr_date(stretch_path, ['20030820']).

ctr_origin(stretch_path, '\\cite{Pesant01}', []).

ctr_usual_name(stretch_path, stretch).

ctr_arguments(
    stretch_path,
    ['VARIABLES'-collection(var-dvar),
     'VALUES'-collection(val-int, lmin-int, lmax-int)]).

ctr_restrictions(
    stretch_path,
    [size('VARIABLES')>0,
     required('VARIABLES', var),
     size('VALUES')>0,
     required('VALUES', [val, lmin, lmax]),
     distinct('VALUES', val),
     'VALUES'^lmin=<'VALUES'^lmax]).

ctr_graph(
    stretch_path,
    ['VARIABLES'],
    2,
    foreach(
        'VALUES',
        ['PATH'>>collection(variables1, variables2),
         'LOOP'>>collection(variables1, variables2)]),
    [variables1^var='VALUES'^val, variables2^var='VALUES'^val],
    [not_in('MIN_NCC', 1, 'VALUES'^lmin-1),
     'MAX_NCC'=<'VALUES'^lmax]).

ctr_example(
    stretch_path,
    stretch_path(
        [var-6],
        [var-6],
        [var-3],
        [var-1],
        [var-1],
        [var-1],
        [var-6],
        [var-6]],
        [[val-1, lmin-2, lmax-4],

```

```
[val-2, lmin-2, lmax-3],  
[val-3, lmin-1, lmax-6],  
[val-6, lmin-2, lmax-2]]) .
```

B.207 strict_lex2

```
ctr_predefined(strict_lex2).  
  
ctr_date(strict_lex2, ['20031016']).  
  
ctr_origin(  
    strict_lex2,  
    '\\cite{FlenerFrischHnichKiziltanMiguelPearsonWalsh02}',  
    []).  
  
ctr_types(strict_lex2, ['VECTOR'-collection(var-dvar)]).  
  
ctr_arguments(strict_lex2, ['MATRIX'-collection(vec-'VECTOR')]).  
  
ctr_restrictions(  
    strict_lex2,  
    [required('VECTOR', var),  
     required('MATRIX', vec),  
     same_size('MATRIX', vec)]).  
  
ctr_example(  
    strict_lex2,  
    strict_lex2(  
        [[vec-[[var-2], [var-2], [var-3]]],  
         [vec-[[var-2], [var-3], [var-1]]]])).
```

B.208 strictly_decreasing

```

ctr_automaton(strictly_decreasing,strictly_decreasing).

ctr_date(strictly_decreasing,['20040814']).

ctr_origin(
    strictly_decreasing,
    'Derived from %c.',
    [strictly_increasing]).

ctr_arguments(
    strictly_decreasing,
    ['VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    strictly_decreasing,
    [size('VARIABLES')>0,required('VARIABLES',var)]).

ctr_graph(
    strictly_decreasing,
    ['VARIABLES'],
    2,
    ['PATH'>>collection(variables1,variables2)],
    [variables1^var>variables2^var],
    ['NARC'=size('VARIABLES')-1]).

ctr_example(
    strictly_decreasing,
    strictly_decreasing([[var-8],[var-4],[var-3],[var-1]])).

strictly_decreasing(A) :-
    strictly_decreasing_signature(A,B),
    automaton(
        B,
        C,
        B,
        0..1,
        [source(s),sink(t)],
        [arc(s,0,s),arc(s,$,t)],
        [],
        [],
        []).

strictly_decreasing_signature([A],[ ]).

```

```
strictly_decreasing_signature([var-A],[var-B|C],[D|E]) :-  
    in(D,0..1),  
    A#=<B#<=>D,  
    strictly_decreasing_signature([var-B|C],E).
```


B.209 strictly_increasing

```

ctr_automaton(strictly_increasing, strictly_increasing) .

ctr_date(strictly_increasing, ['20040814']) .

ctr_origin(strictly_increasing, 'KOALOG', []).

ctr_arguments(
    strictly_increasing,
    ['VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    strictly_increasing,
    [size('VARIABLES')>0, required('VARIABLES', var)]).

ctr_graph(
    strictly_increasing,
    ['VARIABLES'],
    2,
    ['PATH'>>collection(variables1, variables2)],
    [variables1^var<variables2^var],
    ['NARC'=size('VARIABLES')-1]).

ctr_example(
    strictly_increasing,
    strictly_increasing([[var-1], [var-3], [var-4], [var-8]])).

strictly_increasing(A) :-
    strictly_increasing_signature(A, B),
    automaton(
        B,
        C,
        B,
        0..1,
        [source(s), sink(t)],
        [arc(s, 0, s), arc(s, $, t)],
        [],
        [],
        []).

strictly_increasing_signature([A], []).

strictly_increasing_signature([[var-A], [var-B]|C], [D|E]) :-
    in(D, 0..1),
    A#>=B#<=>D,

```

```
strictly_increasing_signature([[var-B] | C], E).
```

B.210 strongly_connected

```
ctr_date(strongly_connected, ['20030820', '20040726']).
```

```
ctr_origin(
  strongly_connected,
  '\\cite{AlthausBockmayrElfKasperJungerMehlhorn02}',
  []).
```

```
ctr_arguments(
  strongly_connected,
  ['NODES'-collection(index-int, succ-svar)]).
```

```
ctr_restrictions(
  strongly_connected,
  [required('NODES', [index, succ]),
   'NODES'^index>=1,
   'NODES'^index=<size('NODES'),
   distinct('NODES', index)]).
```

```
ctr_graph(
  strongly_connected,
  ['NODES'],
  2,
  ['CLIQUE'>>>collection(nodes1, nodes2)],
  [in_set(nodes2^index, nodes1^succ)],
  ['MIN_NSCC'=size('NODES')]).
```

```
ctr_example(
  strongly_connected,
  strongly_connected(
    [[index-1, succ-{2}],
     [index-2, succ-{3}],
     [index-3, succ-{2, 5}],
     [index-4, succ-{1}],
     [index-5, succ-{4}]])).
```

B.211 sum

```

ctr_date(sum, ['20030820', '20040726']).

ctr_origin(sum, '\\cite{Yunes02}.', []).

ctr_arguments(
  sum,
  ['INDEX'-dvar,
   'SETS'-collection(ind-int, set-sint),
   'CONSTANTS'-collection(cst-int),
   'S'-dvar]).

ctr_restrictions(
  sum,
  [size('SETS')>=1,
   required('SETS', [ind, set]),
   distinct('SETS', ind),
   size('CONSTANTS')>=1,
   required('CONSTANTS', cst)]).

ctr_graph(
  sum,
  ['SETS', 'CONSTANTS'],
  2,
  ['PRODUCT'>>collection(sets, constants)],
  ['INDEX'=sets^ind, in_set(constants^key, sets^set)],
  ['SUM'('CONSTANTS', cst)='S']).

ctr_example(
  sum,
  sum(8,
    [[ind-8, set-{2, 3}],
     [ind-1, set-{3}],
     [ind-3, set-{1, 4, 5}],
     [ind-6, set-{2, 4}]],
    [[cst-4], [cst-9], [cst-1], [cst-3], [cst-1]],
    10)).

```

B.212 sum_ctr

```
ctr_date(sum_ctr, ['20030820', '20040807']).  
  
ctr_origin(sum_ctr, 'Arithmetic constraint.', []).  
  
ctr_synonyms(sum_ctr, [constant_sum]).  
  
ctr_arguments(  
    sum_ctr,  
    ['VARIABLES'-collection(var-dvar), 'CTR'-atom, 'VAR'-dvar]).  
  
ctr_restrictions(  
    sum_ctr,  
    [required('VARIABLES', var),  
     in_list('CTR', [=,=\=, <, >=, >, =<])]).  
  
ctr_graph(  
    sum_ctr,  
    ['VARIABLES'],  
    1,  
    ['SELF'>>collection(variables)],  
    ['TRUE'],  
    ['CTR' ('SUM' ('VARIABLES', var), 'VAR')]).  
  
ctr_example(sum_ctr, sum_ctr([[var-1], [var-1], [var-4]], =, 6)).
```

B.213 `sum_of_weights_of_distinct_values`

```

ctr_date(
    sum_of_weights_of_distinct_values,
    ['20030820', '20040726']).

ctr_origin(
    sum_of_weights_of_distinct_values,
    '\\cite{BeldiceanuCarlssonThiel02}',
    []).

ctr_synonyms(sum_of_weights_of_distinct_values, [swdv]).

ctr_arguments(
    sum_of_weights_of_distinct_values,
    ['VARIABLES'-collection(var-dvar),
     'VALUES'-collection(val-int, weight-int),
     'COST'-dvar]).

ctr_restrictions(
    sum_of_weights_of_distinct_values,
    [required('VARIABLES', var),
     required('VALUES', [val, weight]),
     'VALUES'^weight>=0,
     distinct('VALUES', val),
     'COST'>=0]).

ctr_graph(
    sum_of_weights_of_distinct_values,
    ['VARIABLES', 'VALUES'],
    2,
    ['PRODUCT'>>collection(variables, values)],
    [variables^var=values^val],
    ['NSOURCE'=size('VARIABLES'),
     'SUM'('VALUES', weight)='COST']).

ctr_example(
    sum_of_weights_of_distinct_values,
    sum_of_weights_of_distinct_values(
        [[var-1], [var-6], [var-1]],
        [[val-1, weight-5], [val-2, weight-3], [val-6, weight-7]],
        12)).

```

B.214 sum_set

```

ctr_date(sum_set, ['20031001']).

ctr_origin(sum_set, 'H.~Cambazard', []).

ctr_arguments(
    sum_set,
    ['SV'-svar,
     'VALUES'-collection(val-int,coef-int),
     'CTR'-atom,
     'VAR'-dvar]).

ctr_restrictions(
    sum_set,
    [required('VALUES', [val,coef]),
     distinct('VALUES',val),
     'VALUES'^coef>=0,
     in_list('CTR', [=,\=,<,>=,>,<=])]).

ctr_graph(
    sum_set,
    ['VALUES'],
    1,
    ['SELF'>>collection(values)],
    [in_set(values^val,'SV')],
    ['CTR'('SUM'('VALUES',coef),'VAR')]).

ctr_example(
    sum_set,
    sum_set(
        {2,3,6},
        [[val-2,coef-7],
         [val-9,coef-1],
         [val-5,coef-7],
         [val-6,coef-2]],
        =,
        9)).

```

B.215 symmetric_alldifferent

```

ctr_date(symmetric_alldifferent, ['20000128', '20030820']).

ctr_origin(symmetric_alldifferent, '\\cite{Regin99}', []).

ctr_synonyms (
    symmetric_alldifferent,
    [symmetric_alldiff,
     symmetric_alldistinct,
     symm_alldifferent,
     symm_alldiff,
     symm_alldistinct,
     one_factor]).

ctr_arguments (
    symmetric_alldifferent,
    ['NODES' -collection(index-int, succ-dvar)]).

ctr_restrictions (
    symmetric_alldifferent,
    [required('NODES', [index, succ]),
     'NODES'^index>=1,
     'NODES'^index<=size('NODES'),
     distinct('NODES', index),
     'NODES'^succ>=1,
     'NODES'^succ<=size('NODES')]).

ctr_graph (
    symmetric_alldifferent,
    ['NODES'],
    2,
    ['CLIQUE' (=\\=)>>collection(nodes1, nodes2)],
    [nodes1^succ=nodes2^index, nodes2^succ=nodes1^index],
    ['NARC'=size('NODES')]).

ctr_example (
    symmetric_alldifferent,
    symmetric_alldifferent (
        [[index-1, succ-3],
         [index-2, succ-4],
         [index-3, succ-1],
         [index-4, succ-2]])).

```


B.216 symmetric_cardinality

```

ctr_date(symmetric_cardinality, ['20040530']).

ctr_origin(
    symmetric_cardinality,
    'Derived from %c by W.~Kocjan.',
    [global_cardinality]).

ctr_arguments(
    symmetric_cardinality,
    ['VARS'-collection(idvar-int, var-svar, l-int, u-int),
     'VALS'-collection(idval-int, val-svar, l-int, u-int)]).

ctr_restrictions(
    symmetric_cardinality,
    [required('VARS', [idvar, var, l, u]),
     size('VARS')>=1,
     'VARS'^idvar>=1,
     'VARS'^idvar<size('VARS'),
     distinct('VARS', idvar),
     'VARS'^l>=0,
     'VARS'^l<'VARS'^u,
     'VARS'^u<size('VALS'),
     required('VALS', [idval, val, l, u]),
     size('VALS')>=1,
     'VALS'^idval>=1,
     'VALS'^idval<size('VALS'),
     distinct('VALS', idval),
     'VALS'^l>=0,
     'VALS'^l<'VALS'^u,
     'VALS'^u<size('VARS')]).

ctr_graph(
    symmetric_cardinality,
    ['VARS', 'VALS'],
    2,
    ['PRODUCT'>>collection(vars, vals)],
    [#<=>(
        in_set(vars^idvar, vals^val),
        in_set(vals^idval, vars^var)),
     vars^l<card_set(vars^var),
     vars^u>=card_set(vars^var),
     vals^l<card_set(vals^val),
     vals^u>=card_set(vals^val)],
    ['NARC'=size('VARS')*size('VALS')]).

```

```
ctr_example(  
  symmetric_cardinality,  
  symmetric_cardinality(  
    [[idvar-1,var-{3},l-0,u-1],  
     [idvar-2,var-{1},l-1,u-2],  
     [idvar-3,var-{1,2},l-1,u-2],  
     [idvar-4,var-{1,3},l-2,u-3]],  
    [[idval-1,val-{2,3,4},l-3,u-4],  
     [idval-2,val-{3},l-1,u-1],  
     [idval-3,val-{1,4},l-1,u-2],  
     [idval-4,val-{},l-0,u-1]])).
```

B.217 symmetric_gcc

```
ctr_date(symmetric_gcc, ['20030820', '20040530']).
```

```
ctr_origin(
    symmetric_gcc,
    'Derived from %c by W.~Kocjan.',
    [global_cardinality]).
```

```
ctr_synonyms(symmetric_gcc, [sgcc]).
```

```
ctr_arguments(
    symmetric_gcc,
    ['VARS'-collection(idvar-int, var-svar, nocc-dvar),
     'VALS'-collection(idval-int, val-svar, nocc-dvar)]).
```

```
ctr_restrictions(
    symmetric_gcc,
    [required('VARS', [idvar, var, nocc]),
     size('VARS')>=1,
     'VARS'^idvar>=1,
     'VARS'^idvar<size('VARS'),
     distinct('VARS', idvar),
     'VARS'^nocc>=0,
     'VARS'^nocc<size('VALS'),
     required('VALS', [idval, val, nocc]),
     size('VALS')>=1,
     'VALS'^idval>=1,
     'VALS'^idval<size('VALS'),
     distinct('VALS', idval),
     'VALS'^nocc>=0,
     'VALS'^nocc<size('VARS')]).
```

```
ctr_graph(
    symmetric_gcc,
    ['VARS', 'VALS'],
    2,
    ['PRODUCT'>>collection(vars, vals)],
    [#<=>(
        in_set(vars^idvar, vals^val),
        in_set(vals^idval, vars^var)),
     vars^nocc=card_set(vars^var),
     vals^nocc=card_set(vals^val)],
    ['NARC'=size('VARS')*size('VALS')]).
```

```
ctr_example(
```

```
symmetric_gcc,  
symmetric_gcc(  
  [[idvar-1,var-{3},nocc-1],  
   [idvar-2,var-{1},nocc-1],  
   [idvar-3,var-{1,2},nocc-2],  
   [idvar-4,var-{1,3},nocc-2]],  
  [[idval-1,val-{2,3,4},nocc-3],  
   [idval-2,val-{3},nocc-1],  
   [idval-3,val-{1,4},nocc-2],  
   [idval-4,val-{},nocc-0]]).
```

B.218 temporal_path

```

ctr_date(temporal_path, ['20000128', '20030820']).

ctr_origin(temporal_path, 'ILOG', []).

ctr_arguments(
    temporal_path,
    ['NPATH'-dvar,
     -('NODES',
      collection(index-int, succ-dvar, start-dvar, end-dvar))]).

ctr_restrictions(
    temporal_path,
    ['NPATH'>=1,
     'NPATH'=<size('NODES'),
     required('NODES', [index, succ, start, end]),
     size('NODES')>0,
     'NODES'^index>=1,
     'NODES'^index=<size('NODES'),
     distinct('NODES', index),
     'NODES'^succ>=1,
     'NODES'^succ=<size('NODES')]).

ctr_graph(
    temporal_path,
    ['NODES'],
    2,
    ['CLIQUE'>>collection(nodes1, nodes2)],
    [nodes1^succ=nodes2^index,
     nodes1^succ=nodes1^index#\nodes1^end=<nodes2^start,
     nodes1^start=<nodes1^end,
     nodes2^start=<nodes2^end],
    ['MAX_ID'=1, 'NCC'='NPATH', 'NVERTEX'=size('NODES')]).

ctr_example(
    temporal_path,
    temporal_path(
        2,
        [[index-1, succ-2, start-0, end-1],
         [index-2, succ-6, start-3, end-5],
         [index-3, succ-4, start-0, end-3],
         [index-4, succ-5, start-4, end-6],
         [index-5, succ-7, start-7, end-8],
         [index-6, succ-6, start-7, end-9],
         [index-7, succ-7, start-9, end-10]])).

```


B.219 tour

```

ctr_date(tour, ['20030820']).

ctr_origin(
    tour,
    '\\cite{AlthausBockmayrElfKasperJungerMehlhorn02}',
    []).

ctr_synonyms(tour, [atour, cycle]).

ctr_arguments(tour, ['NODES'-collection(index-int, succ-svar)]).

ctr_restrictions(
    tour,
    [size('NODES')>=3,
     required('NODES', [index, succ]),
     'NODES'^index>=1,
     'NODES'^index<=size('NODES'),
     distinct('NODES', index)]).

ctr_graph(
    tour,
    ['NODES'],
    2,
    ['CLIQUE' (=\\=)>>collection(nodes1, nodes2)],
    [#<=>(
        in_set(nodes2^index, nodes1^succ),
        in_set(nodes1^index, nodes2^succ)]),
    ['NARC'=size('NODES')*size('NODES')-size('NODES')]).

ctr_graph(
    tour,
    ['NODES'],
    2,
    ['CLIQUE' (=\\=)>>collection(nodes1, nodes2)],
    [in_set(nodes2^index, nodes1^succ)],
    ['MIN_NSCC'=size('NODES'),
     'MIN_ID'=2,
     'MAX_ID'=2,
     'MIN_OD'=2,
     'MAX_OD'=2]).

ctr_example(
    tour,
    tour(

```

```
[[index-1, succ-{2, 4}],  
 [index-2, succ-{1, 3}],  
 [index-3, succ-{2, 4}],  
 [index-4, succ-{1, 3}]]).
```


B.220 track

```

ctr_date(track, ['20030820']).

ctr_origin(track, '\\cite{Marte01}', []).

ctr_arguments(
    track,
    ['NTRAIL'-int,
     'TASKS'-collection(trail-int,origin-dvar,end-dvar)]).

ctr_restrictions(
    track,
    ['NTRAIL'>0,
     required('TASKS', [trail,origin,end]),
     'TASKS'^trail>0,
     'TASKS'^trail=<'NTRAIL'] ).

ctr_derived_collections(
    track,
    [col(-('TIME_POINTS',
          collection(origin-dvar,end-dvar,point-dvar)),
        [item(
            origin-'TASKS'^origin,
            end-'TASKS'^end,
            point-'TASKS'^origin),
          item(
            origin-'TASKS'^origin,
            end-'TASKS'^end,
            point-'TASKS'^end-1)])]).

ctr_graph(
    track,
    ['TASKS'],
    1,
    ['SELF'>>collection(tasks)],
    [tasks^origin=<tasks^end],
    ['NARC'=size('TASKS')]).

ctr_graph(
    track,
    ['TIME_POINTS', 'TASKS'],
    2,
    ['PRODUCT'>>collection(time_points,tasks)],
    [time_points^end>time_points^origin,
     tasks^origin=<time_points^point,

```

```
time_points^point<tasks^end],
[],
[>>('SUCC',
  [source,
    -(variables,
      col('VARIABLES'-collection(var-dvar),
        [item(var-'TASKS'^trail)]))]]],
[nvalue('NTRAIL',variables)]).

ctr_example(
  track,
  track(
    2,
    [[trail-1,origin-1,end-2],
     [trail-2,origin-1,end-2],
     [trail-1,origin-2,end-4],
     [trail-2,origin-2,end-3],
     [trail-2,origin-3,end-4]]).
```

B.221 tree

```

ctr_date(tree, ['20000128', '20030820']).

ctr_origin(tree, 'N.~Beldiceanu', []).

ctr_arguments(
    tree,
    ['NTREES'-dvar, 'NODES'-collection(index-int, succ-dvar)]).

ctr_restrictions(
    tree,
    ['NTREES'>=0,
     required('NODES', [index, succ]),
     'NODES'^index>=1,
     'NODES'^index<=size('NODES'),
     distinct('NODES', index),
     'NODES'^succ>=1,
     'NODES'^succ<=size('NODES')]).

ctr_graph(
    tree,
    ['NODES'],
    2,
    ['CLIQUE'>>collection(nodes1, nodes2)],
    [nodes1^succ=nodes2^index],
    ['MAX_NSCC'=<1, 'NCC'='NTREES']).

ctr_example(
    tree,
    tree(
        2,
        [[index-1, succ-1],
         [index-2, succ-5],
         [index-3, succ-5],
         [index-4, succ-7],
         [index-5, succ-1],
         [index-6, succ-1],
         [index-7, succ-7],
         [index-8, succ-5]])).

```

B.222 tree_range

```

ctr_date(tree_range, ['20030820', '20040727']).

ctr_origin(tree_range, 'Derived from %c.', [tree]).

ctr_arguments(
    tree_range,
    ['NTREES'-dvar,
     'R'-dvar,
     'NODES'-collection(index-int, succ-dvar)]).

ctr_restrictions(
    tree_range,
    ['NTREES'>=0,
     'R'>=0,
     'R'<size('NODES'),
     required('NODES', [index, succ]),
     'NODES'^index>=1,
     'NODES'^index<size('NODES'),
     distinct('NODES', index),
     'NODES'^succ>=1,
     'NODES'^succ<size('NODES')]).

ctr_graph(
    tree_range,
    ['NODES'],
    2,
    ['CLIQUE'>>collection(nodes1, nodes2)],
    [nodes1^succ=nodes2^index],
    ['MAX_NSCC'=<1, 'NCC'='NTREES', 'RANGE_DRG'='R']).

ctr_example(
    tree_range,
    tree_range(
        2,
        1,
        [[index-1, succ-1],
         [index-2, succ-5],
         [index-3, succ-5],
         [index-4, succ-7],
         [index-5, succ-1],
         [index-6, succ-1],
         [index-7, succ-7],
         [index-8, succ-5]])).

```

B.223 tree_resource

```

ctr_date(tree_resource, ['20030820']).

ctr_origin(tree_resource, 'Derived from %c.', [tree]).

ctr_arguments(
    tree_resource,
    ['RESOURCE'-collection(id-int, nb_task-dvar),
     'TASK'-collection(id-int, father-dvar, resource-dvar)]).

ctr_restrictions(
    tree_resource,
    [required('RESOURCE', [id, nb_task]),
     'RESOURCE'^id>=1,
     'RESOURCE'^id<size('RESOURCE'),
     distinct('RESOURCE', id),
     'RESOURCE'^nb_task>=0,
     'RESOURCE'^nb_task<size('TASK'),
     required('TASK', [id, father, resource]),
     'TASK'^id>size('RESOURCE'),
     'TASK'^id<size('RESOURCE')+size('TASK'),
     distinct('TASK', id),
     'TASK'^father>=1,
     'TASK'^father<size('RESOURCE')+size('TASK'),
     'TASK'^resource>=1,
     'TASK'^resource<size('RESOURCE')]).

ctr_derived_collections(
    tree_resource,
    [col(-('RESOURCE_TASK',
          collection(index-int, succ-dvar, name-dvar)),
        [item(
            index-'RESOURCE'^id,
            succ-'RESOURCE'^id,
            name-'RESOURCE'^id),
         item(
            index-'TASK'^id,
            succ-'TASK'^father,
            name-'TASK'^resource)])]).

ctr_graph(
    tree_resource,
    ['RESOURCE_TASK'],
    2,
    ['CLIQUE'>>collection(resource_task1, resource_task2)],

```

```

[resource_task1^succ=resource_task2^index,
 resource_task1^name=resource_task2^name],
['MAX_NSCC'=<1,
 'NCC'=size('RESOURCE'),
 'NVERTEX'=size('RESOURCE')+size('TASK')]).

ctr_graph(
  tree_resource,
  ['RESOURCE_TASK'],
  2,
  foreach(
    'RESOURCE',
    ['CLIQUE'>>collection(resource_task1,resource_task2)]),
[resource_task1^succ=resource_task2^index,
 resource_task1^name=resource_task2^name,
 resource_task1^name='RESOURCE'^id],
['NVERTEX'='RESOURCE'^nb_task+1]).

ctr_example(
  tree_resource,
  tree_resource(
    [[id-1,nb_task-4],[id-2,nb_task-0],[id-3,nb_task-1]],
    [[id-4,father-8,resource-1],
     [id-5,father-3,resource-3],
     [id-6,father-8,resource-1],
     [id-7,father-1,resource-1],
     [id-8,father-1,resource-1]])).

```

B.224 two_layer_edge_crossing

```

ctr_date(two_layer_edge_crossing, ['20030820']).

ctr_origin(
  two_layer_edge_crossing,
  'Inspired by \\cite{HararySchwenk72}.'.
  []).

ctr_arguments(
  two_layer_edge_crossing,
  ['NCROSS'-dvar,
   'VERTICES_LAYER1'-collection(id-int, pos-dvar),
   'VERTICES_LAYER2'-collection(id-int, pos-dvar),
   'EDGES'-collection(id-int, vertex1-int, vertex2-int)]).

ctr_restrictions(
  two_layer_edge_crossing,
  ['NCROSS'>=0,
   required('VERTICES_LAYER1', [id, pos]),
   'VERTICES_LAYER1'^id>=1,
   'VERTICES_LAYER1'^id<size('VERTICES_LAYER1'),
   distinct('VERTICES_LAYER1', id),
   required('VERTICES_LAYER2', [id, pos]),
   'VERTICES_LAYER2'^id>=1,
   'VERTICES_LAYER2'^id<size('VERTICES_LAYER2'),
   distinct('VERTICES_LAYER2', id),
   required('EDGES', [id, vertex1, vertex2]),
   'EDGES'^id>=1,
   'EDGES'^id<size('EDGES'),
   distinct('EDGES', id),
   'EDGES'^vertex1>=1,
   'EDGES'^vertex1<size('VERTICES_LAYER1'),
   'EDGES'^vertex2>=1,
   'EDGES'^vertex2<size('VERTICES_LAYER2')]).

ctr_derived_collections(
  two_layer_edge_crossing,
  [col(-('EDGES_EXTREMITIES',
        collection(layer1-dvar, layer2-dvar)),
      [item(
        -(layer1,
          'EDGES'^vertex1('VERTICES_LAYER1', pos, id)),
        -(layer2,
          'EDGES'^vertex2('VERTICES_LAYER2', pos, id)))]))].

```

```

ctr_graph(
  two_layer_edge_crossing,
  ['EDGES_EXTREMITIES'],
  2,
  [>>('CLIQUE' (<),
    collection(edges_extremities1,edges_extremities2))],
  [#\/(#\/\(<(edges_extremities1^layer1,
    edges_extremities2^layer1),
    >(edges_extremities1^layer2,
    edges_extremities2^layer2)),
  #\/\(>(edges_extremities1^layer1,
    edges_extremities2^layer1),
    <(edges_extremities1^layer2,
    edges_extremities2^layer2)))],
  ['NARC'='NCROSS'] ).

ctr_example(
  two_layer_edge_crossing,
  two_layer_edge_crossing(
    2,
    [[id-1,pos-1],[id-2,pos-2]],
    [[id-1,pos-3],[id-2,pos-1],[id-3,pos-2]],
    [[id-1,vertex1-2,vertex2-2],
    [id-2,vertex1-2,vertex2-3],
    [id-3,vertex1-1,vertex2-1]]).

```


B.225 two_orth_are_in_contact

```

ctr_automaton(two_orth_are_in_contact,two_orth_are_in_contact).

ctr_date(two_orth_are_in_contact,['20030820','20040530']).

ctr_origin(
    two_orth_are_in_contact,
    'Used for defining %c.',
    [orths_are_connected]).

ctr_types(
    two_orth_are_in_contact,
    ['ORTHOTOPE'-collection(ori-dvar,siz-dvar,end-dvar)]).

ctr_arguments(
    two_orth_are_in_contact,
    ['ORTHOTOPE1'-'ORTHOTOPE','ORTHOTOPE2'-'ORTHOTOPE']).

ctr_restrictions(
    two_orth_are_in_contact,
    [size('ORTHOTOPE')>0,
     require_at_least(2,'ORTHOTOPE',[ori,siz,end]),
     'ORTHOTOPE'^siz>0,
     size('ORTHOTOPE1')==size('ORTHOTOPE2'),
     orth_link_ori_siz_end('ORTHOTOPE1'),
     orth_link_ori_siz_end('ORTHOTOPE2')]).

ctr_graph(
    two_orth_are_in_contact,
    ['ORTHOTOPE1','ORTHOTOPE2'],
    2,
    ['PRODUCT' (=)>>collection(orthotope1,orthotope2)],
    [orthotope1^end>orthotope2^ori,
     orthotope2^end>orthotope1^ori],
    ['NARC'=size('ORTHOTOPE1')-1]).

ctr_graph(
    two_orth_are_in_contact,
    ['ORTHOTOPE1','ORTHOTOPE2'],
    2,
    ['PRODUCT' (=)>>collection(orthotope1,orthotope2)],
    [= (max(0,
            -(max(orthotope1^ori,orthotope2^ori),
                  min(orthotope1^end,orthotope2^end))),
        0)],

```

```

['NARC'=size('ORTHOTOPE1')]).

ctr_example(
    two_orth_are_in_contact,
    two_orth_are_in_contact(
        [[ori-1,siz-3,end-4],[ori-5,siz-2,end-7]],
        [[ori-3,siz-2,end-5],[ori-2,siz-3,end-5]])).

two_orth_are_in_contact(A,B) :-
    two_orth_are_in_contact_signature(A,B,C),
    automaton(
        C,
        D,
        C,
        0..2,
        [source(s),node(z),sink(t)],
        [arc(s,0,s),arc(s,1,z),arc(z,0,z),arc(z,$,t)],
        [],
        [],
        []).

two_orth_are_in_contact_signature([],[],[]).

two_orth_are_in_contact_signature(
    [[ori-A,siz-B,end-C]|D],
    [[ori-E,siz-F,end-G]|H],
    [I|J]) :-
    in(I,0..2),
    B#>0#\F#>0#\C#>E#\G#>A#<=>I#=0,
    B#>0#\F#>0#\C#>E#\G#>A#<=>I#=1,
    two_orth_are_in_contact_signature(D,H,J).

```

B.226 two_orth_column

```

ctr_date(two_orth_column, ['20030820']).

ctr_origin(
    two_orth_column,
    'Used for defining %c.',
    [diffn_column]).

ctr_types(
    two_orth_column,
    ['ORTHOPE' -collection(ori-dvar, siz-dvar, end-dvar)]).

ctr_arguments(
    two_orth_column,
    ['ORTHOPE1' -'ORTHOPE',
     'ORTHOPE2' -'ORTHOPE',
     'N' -int]).

ctr_restrictions(
    two_orth_column,
    [size('ORTHOPE')>0,
     require_at_least(2, 'ORTHOPE', [ori, siz, end]),
     'ORTHOPE'^siz>=0,
     size('ORTHOPE1')=size('ORTHOPE2'),
     orth_link_ori_siz_end('ORTHOPE1'),
     orth_link_ori_siz_end('ORTHOPE2'),
     'N'>0,
     'N'=<size('ORTHOPE1')]).

ctr_graph(
    two_orth_column,
    ['ORTHOPE1', 'ORTHOPE2'],
    2,
    ['PRODUCT' (=)>>collection(orthotope1, orthotope2)],
    [#=>(#/\ (#/\ (#/\ (#/\ (orthotope1^key='N',
                                orthotope1^ori<orthotope2^end),
                                orthotope2^ori<orthotope1^end),
                                orthotope1^siz>0),
                                orthotope2^siz>0),
     #/\ (= (- (min(orthotope1^end, orthotope2^end),
                    max(orthotope1^ori, orthotope2^ori)),
            orthotope1^siz),
            orthotope1^siz=orthotope2^siz))],
    ['NARC'=1]).

```

```
ctr_example(  
  two_orth_column,  
  two_orth_column(  
    [[ori-1,siz-3,end-4],[ori-1,siz-1,end-2]],  
    [[ori-4,siz-2,end-6],[ori-1,siz-3,end-4]],  
    1)).
```

B.227 two_orth_do_not_overlap

```

ctr_automaton(two_orth_do_not_overlap,two_orth_do_not_overlap).

ctr_date(two_orth_do_not_overlap,['20030820','20040530']).

ctr_origin(
    two_orth_do_not_overlap,
    'Used for defining %c.',
    [diffn]).

ctr_types(
    two_orth_do_not_overlap,
    ['ORTHOTOPE'-collection(ori-dvar,siz-dvar,end-dvar)]).

ctr_arguments(
    two_orth_do_not_overlap,
    ['ORTHOTOPE1'-'ORTHOTOPE','ORTHOTOPE2'-'ORTHOTOPE']).

ctr_restrictions(
    two_orth_do_not_overlap,
    [size('ORTHOTOPE')>0,
     require_at_least(2,'ORTHOTOPE',[ori,siz,end]),
     'ORTHOTOPE'^siz>=0,
     size('ORTHOTOPE1')=size('ORTHOTOPE2'),
     orth_link_ori_siz_end('ORTHOTOPE1'),
     orth_link_ori_siz_end('ORTHOTOPE2')]).

ctr_graph(
    two_orth_do_not_overlap,
    ['ORTHOTOPE1','ORTHOTOPE2'],
    2,
    [>>('SYMMETRIC_PRODUCT' (=),
         collection(orthotope1,orthotope2))],
    [orthotope1^end=<orthotope2^ori#\orthotope1^siz=0],
    ['NARC'>=1]).

ctr_example(
    two_orth_do_not_overlap,
    two_orth_do_not_overlap(
        [[ori-2,siz-2,end-4],[ori-1,siz-3,end-4]],
        [[ori-4,siz-4,end-8],[ori-3,siz-3,end-6]]).

two_orth_do_not_overlap(A,B) :-
    two_orth_do_not_overlap_signature(A,B,C),
    automaton(

```

```

C,
D,
C,
0..1,
[source(s),sink(t)],
[arc(s,1,s),arc(s,0,t)],
[],
[],
[]).

```

```
two_orth_do_not_overlap_signature([],[],[]).
```

```

two_orth_do_not_overlap_signature(
  [[ori-A,siz-B,end-C]|D],
  [[ori-E,siz-F,end-G]|H],
  [I|J]) :-
  B#>0#/\F#>0#/\C#>E#/\G#>A#<=>I,
  two_orth_do_not_overlap_signature(D,H,J).

```

B.228 two_orth_include

```

ctr_date(two_orth_include, ['20030820']).

ctr_origin(
    two_orth_include,
    'Used for defining %c.',
    [diffn_include]).

ctr_types(
    two_orth_include,
    ['ORTHOTOPE'-collection(ori-dvar, siz-dvar, end-dvar)]).

ctr_arguments(
    two_orth_include,
    ['ORTHOTOPE1'-'ORTHOTOPE',
     'ORTHOTOPE2'-'ORTHOTOPE',
     'N'-int]).

ctr_restrictions(
    two_orth_include,
    [size('ORTHOTOPE')>0,
     require_at_least(2, 'ORTHOTOPE', [ori, siz, end]),
     'ORTHOTOPE'^siz>=0,
     size('ORTHOTOPE1')=size('ORTHOTOPE2'),
     orth_link_ori_siz_end('ORTHOTOPE1'),
     orth_link_ori_siz_end('ORTHOTOPE2'),
     'N'>0,
     'N'=<size('ORTHOTOPE1')]).

ctr_graph(
    two_orth_include,
    ['ORTHOTOPE1', 'ORTHOTOPE2'],
    2,
    ['PRODUCT' (=)>>collection(orthotope1, orthotope2)],
    [#=>(#/\(#\(\#\(\#\(\(orthotope1^key='N',
                    orthotope1^ori<orthotope2^end),
                    orthotope2^ori<orthotope1^end),
                    orthotope1^siz>0),
                    orthotope2^siz>0),
     #\/(=(-(min(orthotope1^end, orthotope2^end),
                max(orthotope1^ori, orthotope2^ori)),
            orthotope1^siz),
            =(- (min(orthotope1^end, orthotope2^end),
                max(orthotope1^ori, orthotope2^ori)),
            orthotope2^siz)))]],

```

```
['NARC'=1]).  
  
ctr_example(  
  two_orth_include,  
  two_orth_include(  
    [[ori-1,siz-3,end-4],[ori-1,siz-1,end-2]],  
    [[ori-1,siz-2,end-3],[ori-2,siz-3,end-5]],  
    1)).
```


B.229 used_by

```

ctr_date(used_by, ['20000128', '20030820', '20040530']).

ctr_origin(used_by, 'N.~Beldiceanu', []).

ctr_arguments(
    used_by,
    ['VARIABLES1'-collection(var-dvar),
     'VARIABLES2'-collection(var-dvar)]).

ctr_restrictions(
    used_by,
    [size('VARIABLES1')>=size('VARIABLES2'),
     required('VARIABLES1', var),
     required('VARIABLES2', var)]).

ctr_graph(
    used_by,
    ['VARIABLES1', 'VARIABLES2'],
    2,
    ['PRODUCT'>>collection(variables1, variables2)],
    [variables1^var=variables2^var],
    [for_all('CC', 'NSOURCE'>='NSINK'),
     'NSINK'=size('VARIABLES2')]).

ctr_example(
    used_by,
    used_by(
        [[var-1], [var-9], [var-1], [var-5], [var-2], [var-1]],
        [[var-1], [var-1], [var-2], [var-5]])).

```

B.230 used_by_interval

```

ctr_date(used_by_interval, ['20030820']).

ctr_origin(used_by_interval, 'Derived from %c.', [used_by]).

ctr_arguments(
    used_by_interval,
    ['VARIABLES1'-collection(var-dvar),
     'VARIABLES2'-collection(var-dvar),
     'SIZE_INTERVAL'-int]).

ctr_restrictions(
    used_by_interval,
    [size('VARIABLES1')>=size('VARIABLES2'),
     required('VARIABLES1', var),
     required('VARIABLES2', var),
     'SIZE_INTERVAL'>0]).

ctr_graph(
    used_by_interval,
    ['VARIABLES1', 'VARIABLES2'],
    2,
    ['PRODUCT'>>collection(variables1, variables2)],
    [(variables1^var/'SIZE_INTERVAL',
      variables2^var/'SIZE_INTERVAL')],
    [for_all('CC', 'NSOURCE'>='NSINK'),
     'NSINK'=size('VARIABLES2')]).

ctr_example(
    used_by_interval,
    used_by_interval(
        [[var-1], [var-9], [var-1], [var-8], [var-6], [var-2]],
        [[var-1], [var-0], [var-7], [var-7]],
        3)).

```

B.231 used_by_modulo

```

ctr_date(used_by_modulo, ['20030820']).

ctr_origin(used_by_modulo, 'Derived from %c.', [used_by]).

ctr_arguments(
    used_by_modulo,
    ['VARIABLES1'-collection(var-dvar),
     'VARIABLES2'-collection(var-dvar),
     'M'-int]).

ctr_restrictions(
    used_by_modulo,
    [size('VARIABLES1')>=size('VARIABLES2'),
     required('VARIABLES1', var),
     required('VARIABLES2', var),
     'M'>0]).

ctr_graph(
    used_by_modulo,
    ['VARIABLES1', 'VARIABLES2'],
    2,
    ['PRODUCT'>>collection(variables1, variables2)],
    [variables1^var mod 'M'=variables2^var mod 'M'],
    [for_all('CC', 'NSOURCE'>='NSINK'),
     'NSINK'=size('VARIABLES2')]).

ctr_example(
    used_by_modulo,
    used_by_modulo(
        [[var-1], [var-9], [var-4], [var-5], [var-2], [var-1]],
        [[var-7], [var-1], [var-2], [var-5]],
        3)).

```

B.232 used_by_partition

```

ctr_date(used_by_partition, ['20030820']).

ctr_origin(used_by_partition, 'Derived from %c.', [used_by]).

ctr_types(used_by_partition, ['VALUES'-collection(val-int)]).

ctr_arguments(
    used_by_partition,
    ['VARIABLES1'-collection(var-dvar),
     'VARIABLES2'-collection(var-dvar),
     'PARTITIONS'-collection(p-'VALUES')]).

ctr_restrictions(
    used_by_partition,
    [required('VALUES', val),
     distinct('VALUES', val),
     size('VARIABLES1')>=size('VARIABLES2'),
     required('VARIABLES1', var),
     required('VARIABLES2', var),
     required('PARTITIONS', p),
     size('PARTITIONS')>=2]).

ctr_graph(
    used_by_partition,
    ['VARIABLES1', 'VARIABLES2'],
    2,
    ['PRODUCT'>>collection(variables1, variables2)],
    [in_same_partition(
        variables1^var,
        variables2^var,
        'PARTITIONS')],
    [for_all('CC', 'NSOURCE'>='NSINK'),
     'NSINK'=size('VARIABLES2')]).

ctr_example(
    used_by_partition,
    used_by_partition(
        [[var-1], [var-9], [var-1], [var-6], [var-2], [var-3]],
        [[var-1], [var-3], [var-6], [var-6]],
        [[p-[[val-1], [val-3]]],
         [p-[[val-4]]],
         [p-[[val-2], [val-6]]]])).

```

B.233 valley

```

ctr_automaton(valley, valley) .

ctr_date(valley, ['20040530']) .

ctr_origin(valley, 'Derived from %c.', [inflexion]) .

ctr_arguments(
    valley,
    ['N'-dvar, 'VARIABLES'-collection(var-dvar)]) .

ctr_restrictions(
    valley,
    ['N'>=0,
     2*'N'=<max(size('VARIABLES')-1,0),
     required('VARIABLES', var)]) .

ctr_example(
    valley,
    valley(
        1,
        [[var-1],
         [var-1],
         [var-4],
         [var-8],
         [var-8],
         [var-2],
         [var-7],
         [var-1]])) .

valley(A,B) :-
    valley_signature(B,C),
    automaton(
        C,
        D,
        C,
        0..2,
        [source(s), node(u), sink(t)],
        [arc(s,0,s),
         arc(s,1,s),
         arc(s,2,u),
         arc(s,$,t),
         arc(u,0,s,[E+1]),
         arc(u,1,u),
         arc(u,2,u),

```

```
    arc(u, $, t)],  
    [E],  
    [0],  
    [A]).
```

```
valley_signature([], []).
```

```
valley_signature([A], []).
```

```
valley_signature([[var-A], [var-B] | C], [D | E]) :-  
    in(D, 0..2),  
    A#<B#<=>D#=0,  
    A#=B#<=>D#=1,  
    A#>B#<=>D#=2,  
    valley_signature([[var-B] | C], E).
```

B.234 vec_eq_tuple

```
ctr_date(vec_eq_tuple, ['20030820']).

ctr_origin(vec_eq_tuple, 'Used for defining %c.', [in_relation]).

ctr_arguments(
    vec_eq_tuple,
    ['VARIABLES'-collection(var-dvar),
     'TUPLE'-collection(val-int)]).

ctr_restrictions(
    vec_eq_tuple,
    [required('VARIABLES', var),
     required('TUPLE', val),
     size('VARIABLES')=size('TUPLE')]).

ctr_graph(
    vec_eq_tuple,
    ['VARIABLES', 'TUPLE'],
    2,
    ['PRODUCT' (=)>>collection(variables, tuple)],
    [variables^var=tuple^val],
    ['NARC'=size('VARIABLES')]).

ctr_example(
    vec_eq_tuple,
    vec_eq_tuple(
        [[var-5], [var-3], [var-3]],
        [[val-5], [val-3], [val-3]])).
```

B.235 `weighted_partial_alldiff`

```

ctr_date(weighted_partial_alldiff, ['20040814']).

ctr_origin(
    weighted_partial_alldiff,
    '\\cite[page 71]{Thiel04}',
    []).

ctr_synonyms(
    weighted_partial_alldiff,
    [weighted_partial_alldifferent,
     weighted_partial_alldistinct,
     wpa]).

ctr_arguments(
    weighted_partial_alldiff,
    ['VARIABLES'-collection(var-dvar),
     'UNDEFINED'-int,
     'VALUES'-collection(val-int, weight-int),
     'COST'-dvar]).

ctr_restrictions(
    weighted_partial_alldiff,
    [required('VARIABLES', var),
     required('VALUES', [val, weight]),
     in_attr('VARIABLES', var, 'VALUES', val),
     distinct('VALUES', val)]).

ctr_graph(
    weighted_partial_alldiff,
    ['VARIABLES', 'VALUES'],
    2,
    ['PRODUCT' >> collection(variables, values)],
    [variables^var=\='UNDEFINED', variables^var=values^val],
    ['MAX_ID'=<1, 'SUM'('VALUES', weight)='COST']).

ctr_example(
    weighted_partial_alldiff,
    weighted_partial_alldiff(
        [[var-4], [var-0], [var-1], [var-2], [var-0], [var-0]],
        0,
        [[val-0, weight-0],
         [val-1, weight-2],
         [val-2, weight- -1],
         [val-4, weight-7],

```



```
[val-5,weight- -8],  
[val-6,weight-2]],  
8)).
```

Bibliography

- [1] N. Beldiceanu. Global constraints as graph properties on structured network of elementary constraints of the same type. Technical Report T2000-01, Swedish Institute of Computer Science, 2000.
- [2] J.-L. Laurière. A language and a program for stating and solving combinatorial problems. *Artificial Intelligence*, 10:29–127, 1978.
- [3] N. Beldiceanu. Global constraints as graph properties on a structured network of elementary constraints of the same type. In R. Dechter, editor, *Principles and Practice of Constraint Programming (CP'2000)*, volume 1894 of *LNCS*, pages 52–66. Springer-Verlag, 2000. Preprint available as SICS Tech Report T2000-01.
- [4] N. Beldiceanu, M. Carlsson, and T. Petit. Deriving filtering algorithms from constraint checkers. In M. Wallace, editor, *Principles and Practice of Constraint Programming (CP'2004)*, volume 3258 of *LNCS*, pages 107–122. Springer-Verlag, 2004.
- [5] G. Pesant. A regular language membership constraint for finite sequences of variables. In M. Wallace, editor, *Principles and Practice of Constraint Programming (CP'2004)*, volume 3258 of *LNCS*, pages 482–495. Springer-Verlag, 2004.
- [6] H. Simonis, A. Aggoun, N. Beldiceanu, and E. Bourreau. Complex constraint abstraction: Global constraint visualization. lecture. In P. Deransart, M.V. Hermenegildo, and J. Małuszyński, editors, *Analysis and Visualization Tools for Constraint Programming*, volume 1870 of *LNCS*, pages 299–317. Springer-Verlag, 2000.
- [7] G. Rochart and N. Jussien. Explanations for global constraints: instrumenting the *stretch* constraint. Tech. report 03-01-INFO, École des Mines de Nantes, 2003.
- [8] J.N. Hooker and H. Yan. A relaxation for the *cumulative* constraint. In Pascal Van Hentenryck, editor, *Principles and Practice of Constraint Programming (CP'2002)*, volume 2470 of *LNCS*, pages 686–690. Springer-Verlag, 2002. long version at <http://ba.gsia.cmu.edu/jnh/papers.html>.

- [9] M. Bohlin. Desing and implementation of a graph-based constraint model for local search. Licentiate Thesis 27, Mälardalen University, 2004.
- [10] T. Petit, J.-C. Régin, and C. Bessière. Specific filtering algorithms for over-constrained problems. In T. Walsh, editor, *Principles and Practice of Constraint Programming (CP'2001)*, volume 2239 of *LNCS*, pages 451–463. Springer-Verlag, 2001.
- [11] N. Beldiceanu and T. Petit. Cost evaluation of soft global constraints. In Jean-Charles Régin and Michel Rueher, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, volume 3011 of *LNCS*, pages 80–95. Springer-Verlag, 2004.
- [12] W.-J. van Hoesve, G. Pesant, and L.-M. Rousseau. On global warming (softening global constraints). In *Workshop on Soft Constraints*, Toronto, Canada, September 2004.
- [13] L. Euler. Solution d'une question curieuse qui ne parait soumise à aucune analyse. *Mém.Acad.Sci.Berlin*, 15:310–337, 1759.
- [14] H.E. Dudeney. *The Canterbury Puzzles*. Thomas Nelson & Sons, New York, 1919.
- [15] E. Lucas. *Récréations mathématiques*, volume 1-2. Gauthier-Villars, 1882.
- [16] T.P. Kirkman. On a problem in combinatorics. *Cambridge and Dublin Math. J.*, 2:191–204, 1847.
- [17] C. Berge. *Graphes*. Dunod, 1970. In French.
- [18] J.-C. Régin. A filtering algorithm for constraints of difference in CSP. In *12th National Conference on Artificial Intelligence (AAAI-94)*, pages 362–367, 1994.
- [19] J.-C. Régin. Generalized arc consistency for *global cardinality* constraint. In *14th National Conference on Artificial Intelligence (AAAI-96)*, 1996.
- [20] J.-C. Régin. The symmetric *alldiff* constraint. In *16th Int. Joint Conf. on Artificial Intelligence (IJCAI-99)*, 1999.
- [21] J.-C. Régin and M. Rueher. A global constraint combining a *sum* constraint and binary inequalities. In *IJCAI-99 Workshop on Non Binary Constraints*, 1999.
- [22] K. Mehlhorn. Constraint programming and graph algorithms. In U. Montanari, J.D.P. Rolim, and E. Welzl, editors, *27th International Colloquium on Automata, Languages and Programming (ICALP'2000)*, volume 1853 of *LNCS*, pages 571–575. Springer-Verlag, 2000.
- [23] K. Mehlhorn and S. Thiel. Faster algorithms for bound-consistency of the *sortedness* and the *alldifferent* constraint. In *Principles and Practice of Constraint Programming (CP'2000)*, volume 1894 of *LNCS*, pages 306–319. Springer-Verlag, 2000.

- [24] I. Katriel and S. Thiel. Fast bound consistency for the *global cardinality* constraint. In F. Rossi, editor, *Principles and Practice of Constraint Programming (CP'2003)*, volume 2833 of *LNCS*, pages 437–451. Springer-Verlag, 2003.
- [25] N. Beldiceanu, I. Katriel, and S. Thiel. Filtering algorithms for the *same* constraint. In J.-C. Régin and M. Rueher, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimisation Problems (CP-AI-OR 2004)*, volume 3011 of *LNCS*, pages 65–79. Springer-Verlag, 2004.
- [26] W.-J. van Hoeve. A hyper-arc consistency algorithm for the *soft alldifferent* constraint. In M. Wallace, editor, *Principles and Practice of Constraint Programming (CP'2004)*, volume 3258 of *LNCS*, pages 679–689. Springer-Verlag, 2004.
- [27] C.-G. Quimper, A. López-Ortiz, P. van Beek, and A. Golynski. Improved algorithms for the *global cardinality* constraint. In M. Wallace, editor, *Principles and Practice of Constraint Programming (CP'2004)*, volume 3258 of *LNCS*, pages 542–556. Springer-Verlag, 2004.
- [28] R. Dechter and J. Pearl. Network-based heuristics for constraint-satisfaction problems. *Artificial Intelligence*, 34:1–38, 1987.
- [29] C. Berge. *Hypergraphes, Combinatoire des ensembles finis*. Dunod, 1987. In French.
- [30] S. Skiena. *Implementing Discrete Mathematics. Combinatoric and Graph Theory with Mathematica*. Addison-Wesley, 1990.
- [31] M. Gondran and M. Minoux. *Graphs and Algorithms*. Wiley, New York, 2nd revised edition edition, 1984.
- [32] P. Van Hentenryck and J.-P. Carillon. Generality vs. specificity: an experience with AI and OR techniques. In *National Conference on Artificial Intelligence (AAAI-88)*, 1988.
- [33] N. Beldiceanu. Pruning for the *minimum* constraint family and for the *number of distinct values* constraint family. In T. Walsh, editor, *Principles and Practice of Constraint Programming (CP'2001)*, volume 2239 of *LNCS*, pages 211–224. Springer-Verlag, 2001. Preprint available as SICS Tech Report T2000-10.
- [34] S. Bourdais, P. Galinier, and G. Pesant. HIBISCUS: A constraint programming application to staff scheduling in health care. In F. Rossi, editor, *Principles and Practice of Constraint Programming (CP'2003)*, volume 2833 of *LNCS*, pages 153–167. Springer-Verlag, 2003.
- [35] M. Maher. Analysis of a *global contiguity* constraint. In *Workshop on Rule-Based Constraint Reasoning and Programming*, 2002. held along CP-2002.

- [36] A. Frisch, B. Hnich, Z. Kızıltan, I. Miguel, and T. Walsh. Global Constraints for lexicographic orderings. In Pascal Van Hentenryck, editor, *Principles and Practice of Constraint Programming (CP'2002)*, volume 2470 of *LNCS*, pages 93–108. Springer-Verlag, 2002.
- [37] N. Beldiceanu and E. Contejean. Introducing global constraints in CHIP. *Mathl. Comput. Modelling*, 20(12):97–123, 1994.
- [38] M. Dincbas, P. Van Hentenryck, H. Simonis, T. Graf A. Aggoun, and F. Berthier. The Constraint Logic Programming Language CHIP. In *Int. Conf. on Fifth Generation Computer Systems (FGCS'88)*, pages 693–702, Tokyo, Japan, 1988.
- [39] F. Laburthe. Choco: implementing a cp kernel. In *CP'00 Workshop on Techniques for Implementing Constraint programming Systems (TRICS)*, 2000.
- [40] PLATON team. *Eclair*. Thales R & T, Orsay, France, v8.0 edition, 2003. technical report 61 364.
- [41] A.M. Cheadle, W. Harvey, A.J. Sadler, J. Schimpf, K. Shen, and M.G. Wallace. Eclipse: An introduction. Technical Report 03-1, IC-Parc, Imperial College London, 2003.
- [42] J.-F. Puget. A c++ implementation of clp. In *Second Singapore International Conference on Intelligent Systems (SPICIS)*, pages 256–261, Singapore, November 1994.
- [43] G. Smolka. Constraints in Oz. *ACM Computing Surveys*, 28(4), 1996.
- [44] M. Carlsson, G. Ottosson, and B. Carlson. An open-ended finite domain constraint solver. In H. Glaser, P. Hartel, and H. Kuchen, editors, *Programming Languages: Implementations, Logics, and Programming (PLILP'97)*, volume 1292 of *LNCS*, pages 191–206, Southampton, 1997. Springer-Verlag.
- [45] COSYTEC. *CHIP Reference Manual*, release 5.1 edition, 1997.
- [46] Mats Carlsson et al. *SICStus Prolog User's Manual*. Swedish Institute of Computer Science, 3.10 edition, January 2003. <http://www.sics.se/sicstus/>.
- [47] E. Poder, N. Beldiceanu, and E. Sanlaville. Computing a lower approximation of the compulsory part of a task with varying duration and varying resource consumption. *European Journal of Operational Research*, 153:239–254, 2004.
- [48] N. Beldiceanu and E. Poder. Cumulated profiles of minimum and maximum resource utilisation. In *Ninth Int. Conf. on Project Management and Scheduling*, 2004.
- [49] J.-C. Régim and M. Rueher. *inequality-sum*: A global constraint capturing the objective function. *RAIRO Operations Research*, 2005. To appear.

- [50] Y. Caseau and F. Laburthe. Solving small TSPs with constraints. In Lee Naish, editor, *Fourteenth International Conference on Logic Programming (ICLP'97)*, pages 316–330. MIT Press, 1997.
- [51] C. Bessière, E. Hebrard, B. Hnich, Z. Kızıltan, and T. Walsh. The *range* and *roots* constraints: Specifying counting and occurrence problems. In *19th Int. Joint Conf. on Artificial Intelligence (IJCAI-05)*, 2005.
- [52] R. Dechter and J. Pearl. Tree clustering for constraint networks. *Artificial Intelligence*, 38:353–366, 1989.
- [53] R. Fagin. Degrees of acyclicity for hypergraphs and relational database schemes. *JACM*, 30:514–550, July 1983.
- [54] C. Bessière, E. Hebrard, B. Hnich, Z. Kızıltan, and T. Walsh. Filtering algorithms for the *nvalue* constraint. In Romand Barták and Michela Milano, editors, *International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CP-AI-OR'05)*, Lecture Notes in Computer Science, Prague, Czech Republic, may 2005. Springer Verlag.
- [55] P. Turán. On an extremal problem in graph theory. *Mat. Fiz. Lapok*, 48:436–452, 1941. In Hungarian.
- [56] A.G. Frutos, Q. Liu, A.J. Thiel, A.M.W. Sanner, A.E. Condon, L.M. Smith, and R.M. Corn. Demonstration of a word design strategy for DNA computing on surfaces. *Nucleic Acids Research*, 25:4748–4757, 1997.
- [57] J.-C. Régim. The global minimum distance constraint. Technical report, ILOG, 1997.
- [58] J.-L. Laurière. *Un langage et un programme pour énoncer et résoudre des problèmes combinatoires*. Thèse de doctorat d'état, Université Paris 6, May 1976. In French.
- [59] M.-C. Costa. Persistency in maximum cardinality bipartite matchings. *Operation Research Letters*, 15:143–149, 1994.
- [60] M. Leconte. A bounds-based reduction scheme for constraints of difference. In *CP'96, Second International Workshop on Constraint-based Reasoning*, pages 19–28, Key West, FL, USA, 1996.
- [61] N. Bleuzen-Guernalec and A. Colmerauer. Narrowing a block of sortings in quadratic time. In G. Smolka, editor, *Principles and Practice of Constraint Programming (CP'97)*, volume 1330 of *LNCS*, pages 2–16. Springer-Verlag, 1997.
- [62] J.-F. Puget. A fast algorithm for the bound consistency of *alldiff* constraints. In *15th National Conference on Artificial Intelligence (AAAI-98)*, pages 359–366. AAAI Press, 1990.

- [63] A. Lopez-Ortiz, C.-G. Quimper, J. Tromp, and P. van Beek. A fast and simple algorithm for bounds consistency of the alldifferent constraint. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI'2003)*, pages 245–250, 2003.
- [64] A. Frisch, C. Jefferson, and I. Miguel. Constraints for breaking more row and column symmetries. In F. Rossi, editor, *Principles and Practice of Constraint Programming (CP'2003)*, volume 2833 of *LNCS*, pages 318–332. Springer-Verlag, 2003.
- [65] N. Beldiceanu and M. Carlsson. Revisiting the *cardinality* operator and introducing the *cardinality-path* constraint family. In P. Codognet, editor, *Int. Conf. on Logic Programming (ICLP'2001)*, volume 2237 of *LNCS*, pages 59–73. Springer-Verlag, 2001. Preprint available as SICS Tech Report T2000-11A.
- [66] S. Martello and P. Toth. *Knapsack problems. Algorithms and Computer Implementations*. Interscience Series in Discrete Mathematics and Optimization. Wiley, 1990.
- [67] A. Aggoun and N. Beldiceanu. Extending CHIP in order to solve complex scheduling and placement problems. *Mathl. Comput. Modelling*, 17(7):57–73, 1993.
- [68] P. Shaw. A constraint for bin packing. In M. Wallace, editor, *Principles and Practice of Constraint Programming (CP'2004)*, volume 3258 of *LNCS*, pages 648–662. Springer-Verlag, 2004.
- [69] M. Müller-Hannemann, W. Stille, and K. Weihe. Patterns of usage for global constraints: A case study based on the bin-packing constraint. Research report, TU Darmstadt, 2003.
- [70] M. Müller-Hannemann, W. Stille, and K. Weihe. Evaluating the bin-packing constraint, part i: Overview of the algorithmic approach. Research report, TU Darmstadt, 2003.
- [71] M. Müller-Hannemann, W. Stille, and K. Weihe. Evaluating the bin-packing constraint, part ii: An adaptive rounding problem. Research report, TU Darmstadt, 2003.
- [72] M. Müller-Hannemann, W. Stille, and K. Weihe. Evaluating the bin-packing constraint, part iii: Joint evaluation with concave constraints. Research report, TU Darmstadt, 2003.
- [73] F. Pachet and P. Roy. Automatic generation of music programs. In *Principles and Practice of Constraint Programming (CP'99)*, volume 1713 of *LNCS*, pages 331–345. Springer-Verlag, 1999.
- [74] E. Althaus, A. Bockmayr, M. Elf, T. Kasper, M. Jünger, and K. Mehlhorn. SCIL—symbolic constraints in integer linear programming. In *10th European Symposium on Algorithms (ESA'02)*, volume 2461 of *LNCS*, pages 75–87. Springer-Verlag, September 2002.

- [75] J.A. Shufet and H.J. Berliner. Generating hamiltonian circuits without backtracking from errors. *Theoretical Computer Science*, 1994.
- [76] E.Ya. Grinberg. Plane homogeneous graphs of degree three without hamiltonian circuits. *Latv. Mat. Ezhegodnik*, 1968.
- [77] G. Laporte, A. Asef-Vaziri, and C. Sriskandarajah. Some applications of the generalized travelling salesman problem. *J. of the Operational Research Society*, 47:1461–1467, 1996.
- [78] T. Fahle. Cost based filtering vs. upper bounds for maximum clique. In *4th Int. Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR'03)*, Le Croisic, France, 2002.
- [79] J.-C. Régin. Using constraint programming to solve the maximum clique problem. In F. Rossi, editor, *Principles and Practice of Constraint Programming (CP'2003)*, volume 2833 of *LNCS*, pages 634–648. Springer-Verlag, 2003.
- [80] J.-C. Régin and C. Gomes. The *cardinality matrix* constraint. In M. Wallace, editor, *Principles and Practice of Constraint Programming (CP'2004)*, volume 3258 of *LNCS*, pages 572–587. Springer-Verlag, 2004.
- [81] L.R. Ford Jr. and D.R. Fulkerson. *Flows in Networks*. Princeton University Press, 1962.
- [82] H. Simonis. Channel routing seen as a constraint problem. Tech. Report TR-LP-51, ECRC, 1990.
- [83] N.-F. Zhou. Channel routing with constraint logic programming and delay. In *9th Int. Conf. on Industrial Applications of AI*, pages 217–231. Gordon and Breach Science Publishers, 1996.
- [84] T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to algorithms*. MIT Press, fifteenth edition, 1990.
- [85] A. Lahrichi. Scheduling: the notions of hump, compulsory parts and their use in cumulative problems. *C. R. Acad. Sci., Paris*, 294:209–211, Feb 1982.
- [86] J. Erschler and P. Lopez. Energy-based approach for task scheduling under time and resources constraints. In *2nd International Workshop on Project Management and Scheduling*, pages 115–121, Compiègne, France, June 1990.
- [87] Y. Caseau and F. Laburthe. Cumulative scheduling with task intervals. In *Joint International Conference and Symposium on Logic Programming (JICSLP'96)*. MIT Press, 1996.
- [88] H. Samet. *The design and analysis of spatial data structures*. Addison-Wesley, 1989.

- [89] N. Beldiceanu and M. Carlsson. A new multi-resource *cumulatives* constraint with negative heights. In P. Van Hentenryck, editor, *Principles and Practice of Constraint Programming (CP'2002)*, volume 2470 of *LNCS*, pages 63–79. Springer-Verlag, 2002. Preprint available as SICS Tech Report T2001-11.
- [90] F. Fages and A. Lal. A global constraint for cutset problems. In *5th Int. Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR'03)*, Montréal, 2003.
- [91] H. Levy and D.W. Low. A contraction algorithm for finding small cycle cutsets. *J. of Algorithms*, 9:470–493, 1988.
- [92] E.L. Lloyd, M.L. Soffa, and C.C. Wang. On locating minimum feedback vertex sets. *J. of Computer and System Science*, 37:292–311, 1988.
- [93] E. Bourreau. *Traitement de contraintes sur les graphes en programmation par contraintes*. PhD thesis, University Paris 13, March 1999. In French.
- [94] M. Labbé, G. Laporte, and I. Rodríguez-Martín. Path, tree and cycle location. In *Fleet Management and Logistics*, pages 187–204. Kluwer Academic Publishers, 1998.
- [95] R. Szymanek. *Constraint-Driven Design Space Exploration for Memory-Dominated Embedded Systems*. PhD thesis, Lund University, June 2004.
- [96] R. Szymanek and K. Kuchcinski. A constructive algorithm for memory-aware task assignment and scheduling. In *Proceedings of the Ninth International Symposium on Hardware/Software Codesign*, Copenhagen, 2001.
- [97] N. Beldiceanu and M. Carlsson. Sweep as a generic pruning technique applied to the non-overlapping rectangles constraints. In T. Walsh, editor, *Principles and Practice of Constraint Programming (CP'2001)*, volume 2239 of *LNCS*, pages 377–391. Springer-Verlag, 2001.
- [98] N. Beldiceanu, Q. Guo, and S. Thiel. Non-overlapping constraints between convex polytopes. In T. Walsh, editor, *Principles and Practice of Constraint Programming (CP'2001)*, volume 2239 of *LNCS*, pages 392–407. Springer-Verlag, 2001. Preprint available as SICS Tech Report T2001-12.
- [99] C. Ribeiro and M.A. Carravilla. A global constraint for nesting problems. In J.-C. Régim and M. Rueher, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimisation Problems (CP-AI-OR 2004)*, volume 3011 of *LNCS*, pages 256–270. Springer-Verlag, 2004.
- [100] C.J. Bouwkamp and Duijvestijn. Catalogue of simple perfect squared squares of orders 21 through 25. Research report 92-WSK-03, Eindhoven University of Technology, November 1992.
- [101] I. Gambini. *Quant aux carrés carrelés*. PhD thesis, University Aix-Marseille II, December 1999. In French.

- [102] I. Gambini. A method for cutting squares into distinct squares. *Discrete Applied Mathematics*, 98(1-2):65–80, 1999.
- [103] F. Focacci. *Solving Combinatorial Optimization Problems in Constraint Programming*. PhD thesis, University of Ferrara, 2001.
- [104] W.-J. van Hoeve. *Operations Research Techniques in Constraint Programming*. PhD thesis, University of Amsterdam, CWI, 2005.
- [105] M.L. Ginsberg and W.D. Harvey. Limited discrepancy search. In C.S. Mellish, editor, *14th Int. Joint Conf. on Artificial Intelligence (IJCAI-95)*, volume 1, pages 607–615. Morgan Kaufmann, 1995.
- [106] N. Beldiceanu, M. Carlsson, and S. Thiel. Cost-filtering algorithms for the two sides of the *sum of weights of distinct values* constraint. Technical Report T2002-14, Swedish Institute of Computer Science, 2002.
- [107] J. Carlier. One machine problem. *European Journal of Operational Research*, 11:42–47, 1982.
- [108] P. Baptiste, C. Le Pape, and L. Peridy. Global constraints for partial csp: A case-study of resource and due date constraints. In M. Maher and J.-F. Puget, editors, *Principles and Practice of Constraint Programming (CP'98)*, volume 1520 of *LNCS*, pages 87–101. Springer-Verlag, 1998.
- [109] P. Vilím. $O(n \log n)$ filtering algorithms for unary resource constraint. In J.-C. Régin and M. Rueher, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimisation Problems (CP-AI-OR 2004)*, volume 3011 of *LNCS*, pages 335–347. Springer-Verlag, 2004.
- [110] L. Péridy and D. Rivreau. An $O(n \log n)$ stable algorithm for immediate selections adjustments. Kluwer, 2005. To appear.
- [111] P. Refalo. Linear formulation of constraint programming models and hybrid solvers. In *Principles and Practice of Constraint Programming (CP'2000)*, volume 1894 of *LNCS*. Springer-Verlag, 2000.
- [112] G. Ottosson, E. Thorsteinsson, and J.N. Hooker. Mixed global constraints and inference in hybrid IP-CLP solvers. In *CP'99 Post-Conference Workshop on Large-Scale Combinatorial Optimization and Constraints*, pages 57–78, 1999.
- [113] P. Van Hentenryck. *Constraint Satisfaction in Logic Programming*. MIT Press, 1989.
- [114] C. Bessière, E. Hebrard, B. Hnich, and T. Walsh. The tractability of global constraints. In M. Wallace, editor, *Principles and Practice of Constraint Programming (CP'2004)*, volume 3258 of *LNCS*, pages 716–720. Springer-Verlag, 2004.

- [115] C.-G. Quimper, P. van Beek, A. López-Ortiz, A. Golynski, and S.B. Sadjad. An efficient bounds consistency algorithm for the *global cardinality* constraint. In F. Rossi, editor, *Principles and Practice of Constraint Programming (CP'2003)*, volume 2833 of *LNCS*, pages 600–614. Springer-Verlag, 2003.
- [116] I.Katriel and S.Thiel. Complete bound consistency for the global cardinality constraint. *Constraints*, 10(3), 2005.
- [117] J.-C. Régin. Arc consistency for global cardinality constraints with costs. In J. Jaffar, editor, *Principles and Practice of Constraint Programming (CP'99)*, volume 1713 of *LNCS*, pages 390–404. Springer-Verlag, 1999.
- [118] S.W. Golomb. How to number a graph. In R.C. Read, editor, *Graph Theory and Computing*, pages 23–37. Academic Press, New York, 1972.
- [119] J.B. Shearer. Golomb rulers. <http://www.research.ibm.com/people/s/shearer/grule.html>.
- [120] B.M. Smith, K. Stergiou, and T. Walsh. Modelling the golomb ruler problem. In *IJCAI-99 Workshop on Non Binary Constraints*, 1999.
- [121] Y.C. Law and J.H.M. Lee. Global constraints for integer and set value precedence. In M. Wallace, editor, *Principles and Practice of Constraint Programming (CP'2004)*, volume 3258 of *LNCS*, pages 362–376. Springer-Verlag, 2004.
- [122] X. Cousin. *Application of Constraint Logic Programming on Timetable Problem*. PhD thesis, INRIA, June 1993. In French.
- [123] P. Flener, A. Frisch, B. Hnich, Z. Kızıltan, I. Miguel, J. Pearson, and T. Walsh. Breaking row and column symmetries in matrix models. In F. Rossi, editor, *Principles and Practice of Constraint Programming (CP'2003)*, volume 2833 of *LNCS*, pages 462–476. Springer-Verlag, 2003.
- [124] A. Lubiw. Doubly lexical orderings of matrices. In *Proceedings of the 17th Annual Association for Computing Machinery Symposium on Theory of Computing (STOC-85)*, pages 396–404. ACM Press, 1985.
- [125] A. Lubiw. Doubly lexical orderings of matrices. *SIAM Journal on Computing*, 16(5):854–879, October 1987.
- [126] M. Carlsson and N. Beldiceanu. Arc-consistency for a *chain of lexicographic ordering* constraints. Technical Report T2002-18, Swedish Institute of Computer Science, 2002.
- [127] A. Frisch, B. Hnich, Z. Kızıltan, I. Miguel, and T. Walsh. Multiset ordering constraints. In *18th Int. Joint Conf. on Artificial Intelligence (IJCAI-2003)*, 1999.
- [128] M. Carlsson and N. Beldiceanu. Revisiting the *lexicographic ordering* constraint. Technical Report T2002-17, Swedish Institute of Computer Science, 2002.

- [129] Z. Kızıltan. *Symmetry Breaking Ordering Constraints*. PhD thesis, Uppsala University, March 2004.
- [130] R. Sedgewick and O. Flajolet. *An introduction to the analysis of algorithms*. Addison-Wesley, 1996.
- [131] F. Focacci, A. Lodi, and M. Milano. Cost-based domain filtering. In *Principles and Practice of Constraint Programming (CP'99)*, volume 1713 of *LNCS*, pages 189–203. Springer-Verlag, 1999.
- [132] M. Sellman. An arc consistency algorithm for the *minimum weight all different* constraint. In P. Van Hentenryck, editor, *Principles and Practice of Constraint Programming (CP'2002)*, volume 2470 of *LNCS*, pages 744–749. Springer-Verlag, 2002.
- [133] J.-C. Régin. *Développement d'outils algorithmiques pour l'Intelligence Artificielle*. PhD thesis, University of Montpellier II, 1995. In French.
- [134] I. Gent, P. Prosser, B. Smith, and W. Wei. Supertree construction with constraint programming. In F. Rossi, editor, *Principles and Practice of Constraint Programming (CP'2003)*, volume 2833 of *LNCS*, pages 837–841. Springer-Verlag, 2003.
- [135] J. Jackson. *Rational amusements for winter evenings*. Longman, London, 1821.
- [136] N. Beldiceanu and E. Poder. The *period* constraint. In B. Demoen, editor, *Int. Conf. on Logic Programming (ICLP'2004)*, *LNCS*. Springer-Verlag, 2004.
- [137] S.W. Golomb. *Polyominoes*. Scribners, New York, 1965.
- [138] D. Gale. A theorem on flows in networks. *Pacific J. Math.*, 7:1073–1082, 1957.
- [139] W.J. Older, G.M. Swinkels, and M.H. Van Emden. Getting to the real problem: Experience with BNR Prolog in OR. In *3rd Int. Conf. on the Practical Application of Prolog (PAP'95)*, pages 465–478. Alinmead Software Ltd., 1995.
- [140] Z. Kızıltan and T. Walsh. Constraint programming with multisets. In *Workshop on Symmetry on Constraint Satisfaction Problems (SymCon-02)*, 2002. held along CP-2002.
- [141] N. Beldiceanu, I. Katriel, and S. Thiel. Filtering algorithms for the *same* and *usedby* constraints. Research Report 2004/1/001, MPI, 2004.
- [142] N. Beldiceanu, I. Katriel, and S. Thiel. Filtering algorithms for the *same* and *usedby* constraints. *Archives of Control Sciences, Special Issue on constraint programming for decision and control*. To appear.
- [143] N. Beldiceanu, I. Katriel, and S. Thiel. Gcc-like restrictions on the *same* constraint. In *Recent Advances in Constraints (CSCLP 2004)*, volume 3419 of *LNAI*. Springer-Verlag, 2004.

- [144] C. Flamm, I.L. Hofacker, and P.F. Stadler. RNA in silico: The computational biology of RNA secondary structures. *Adv. Complex Syst.*, 2:5–90, 1999.
- [145] S. Mittal and B. Falkenhainer. Dynamic constraint satisfaction problems. In *8th National Conference on Artificial Intelligence (AAAI-90)*, pages 25–32. AAAI Press, 1990.
- [146] J.-C. Régin and J.-F. Puget. A filtering algorithm for global sequencing constraints. In G. Smolka, editor, *Principles and Practice of Constraint Programming (CP'97)*, volume 1330 of *LNCS*, pages 32–46. Springer-Verlag, 1997.
- [147] J. Zhou. A permutation-based approach for solving the job-shop problem. *Constraints*, 2(2):185–213, 1997.
- [148] G. Pesant. A filtering algorithm for the *stretch* constraint. In T. Walsh, editor, *Principles and Practice of Constraint Programming (CP'2001)*, volume 2239 of *LNCS*, pages 183–195. Springer-Verlag, 2001.
- [149] L. Hellsten, G. Pesant, and P. van Beek. A domain consistency algorithm for the *stretch* constraint. In M. Wallace, editor, *Principles and Practice of Constraint Programming (CP'2004)*, volume 3258 of *LNCS*, pages 290–304. Springer-Verlag, 2004.
- [150] Talys H. Yunes. On the *sum* constraint: Relaxation and applications. In P. Van Hentenryck, editor, *Principles and Practice of Constraint Programming (CP'2002)*, volume 2470 of *LNCS*, pages 80–92. Springer-Verlag, 2002.
- [151] G. Pesant and P. Soriano. An optimal strategy for the constrained cycle cover problem. CRT Pub. 98-45, CRT, Montréal, December 1998.
- [152] M. Henz, T. Müller, and S. Thiel. Global constraints for round robin tournament scheduling. *European Journal of Operations Research*, 153(1):92–101, Feb 2004.
- [153] M. Trick. Integer and constraint programming approaches for round robin tournament scheduling. In E.K. Burke and P. De Causmaecker, editors, *Practice and Theory of Automated Timetabling IV, 4th International Conference, PATAT 2002, Gent, Belgium, August 21-23, 2002, Selected Revised Papers*, volume 2740 of *LNCS*, pages 63–77. Springer-Verlag, 2003.
- [154] W. Kocjan and P. Kreuger. Filtering methods for *symmetric cardinality* constraint. In J.-C. Régin and M. Rueher, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimisation Problems (CP-AI-OR 2004)*, volume 3011 of *LNCS*, pages 200–208. Springer-Verlag, 2004.
- [155] M. Marte. A global constraint for parallelizing the execution of task sets in non-preemptive scheduling. In *CP'2001 Doctoral Programme*, 2001.

- [156] N. Beldiceanu, P. Flener, and X. Lorca. The *tree* constraint. In Romand Barták and Michela Milano, editors, *International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CP-AI-OR'05)*, Lecture Notes in Computer Science, Prague, Czech Republic, may 2005. Springer Verlag.
- [157] F. Harary and A.J. Schwenk. A new crossing number for bipartite graphs. *Utilitas Math.*, 1:203–209, 1972.
- [158] G. Di Battista, P. Eades, R. Tamassia, and I.G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice-Hall, Englewood Cliffs, NJ, 1999.
- [159] M.R. Garey and D.S. Johnson. Crossing number is np-complete. *SIAM J. Algebraic Discrete Methods*, 4:312–316, 1983.
- [160] S. Thiel. *Efficient Algorithms for Constraint Propagation and for Processing Tree Descriptions*. PhD thesis, Saarlandes University, 2004.

Index

acyclic, **62**, 195, 215, 230, 274, 278, 281, 285, 290, 300, 304, 333, 337, 339, 341, 348, 356, 404, 407

Aggoun A., 264, 362

alignment, **63**, 713

all different, **62**, 179, 180, 184, 188, 192, 195, 198, 811, 815, 883, 948

all_differ_from_at_least_k_pos, 13, 70, 80, 81, 93, 109, **172**, 422, 424

all_min_dist, 80, 85, 108, **174**

all_null_intersect, **180**

ALL_VERTICES, **47**, 310

alldiff, **176**

alldiff_between_sets, **180**

alldiff_except_0, **182**

alldiff_interval, **186**

alldiff_modulo, **190**

alldiff_on_intersection, **194**

alldiff_partition, **198**

alldiff_same_value, **200**

alldifferent, 5, 13, 27, 43, 51–54, 62, 65, 67, 71, 72, 81, 82, 87, 93, 95, 97, 108, 112, 161, 174, **176**, 180, 182, 184, 186, 188, 190, 192, 194, 195, 198, 200, 307, 312, 387, 419, 436, 438, 482–484, 498, 504, 510, 582, 668, 700, 756, 758, 782, 784, 810, 811, 814, 815, 883, 948

alldifferent_between_sets, 62, 71, 76, 81, 95, 112, **180**, 490

alldifferent_except_0, 62, 65, 67, 88, 95, 99, 108, 178, 179, **182**, 948

alldifferent_interval, 62, 65, 67, 88, 95, 108, **186**

alldifferent_modulo, 62, 65, 67, 92, 95, 108, **190**

alldifferent_on_intersection, 62, 65, 67, 70, 75, 77, 93, 108, 179, **194**, 333, 702, 764

alldifferent_partition, 62, 95, 96, 108, **198**, 543

alldifferent_same_value, 30, 36, 65, 67, 98, **200**

alldistinct, **176**

alldistinct_between_sets, **180**

alldistinct_except_0, **182**

- alldistinct_interval, **186**
- alldistinct_modulo, **190**
- alldistinct_on_intersection, **194**
- alldistinct_partition, **198**
- alldistinct_same_value, **200**
- allperm, 89, 91, 95, 98, 105, **204**, 580, 862
- alpha-acyclic constraint network(2), **63**, 207, 210, 213, 215, 220, 244, 248, 352, 356, 424, 494, 523, 530, 789
- alpha-acyclic constraint network(3), **63**, 523, 530, 577
- alpha-acyclic constraint network(4), **64**, 624, 642
- Althaus E., 306, 578, 726, 868, 896
- among, 11, 29, 51–53, 63, 65, 68, 79, 108, **206**, 210, 212, 213, 215, 218, 220, 223, 225, 236, 244, 248, 274, 278, 352, 356, 435, 494, 498, 537, 630, 646, 789
- among_diff_0, 63, 65, 68, 79, 88, 108, 207, **208**, 700
- among_interval, 63, 65, 68, 79, 88, 108, **212**
- among_low_up, 62, 63, 65, 68, 70, 79, 93, 108, **214**, 223, 391, 392, 562, 789
- among_modulo, 63, 65, 68, 79, 92, 108, **218**
- among_seq, 80, 87, 100, 103, 215, **222**, 789, 791
- apartition, **64**, 290
- arith, 65, 68, 80, 82, 108, **224**, 230, 232
- arith_or, 62, 65, 68, 70, 80, 93, 108, **228**
- arith_sliding, 44, 65, 68, 80, 87, 100, 103, 224, **232**
- arithmetic constraint, **64**, 748, 750, 875, 880
- array constraint, **64**, 459, 462, 465, 469, 475, 477
- Asef-Vaziri A., 310
- assign_and_counts, 64, 65, 67, 75, 81, 112, **234**, 238, 356
- assign_and_nvalues, 64, 94, 112, **238**, 704, 706
- assignment, **64**, 236, 240, 250, 254, 258, 261, 266, 274, 278, 498, 501, 504, 553, 562, 566, 630, 648, 650, 668, 762, 877, 888, 891, 948
- assignment, **568**
- at least, **65**, 244, 274

at most, **65**, 248, 278, 281

atleast, 10, 11, 63, 65, 68, 108, 236, **242**, 248, 492, 494

atmost, 63, 65, 68, 108, 236, 244, **246**, 391, 492, 494

atour, **306**, 307, **896**

automaton, **65**, 179, 184, 188, 192, 195, 201, 207, 210, 213, 215, 220, 225, 230, 232, 236, 244, 248, 250, 254, 258, 266, 274, 278, 285, 290, 300, 316, 352, 356, 365, 404, 407, 411, 416, 424, 438, 449, 453, 459, 462, 465, 469, 475, 477, 494, 498, 506, 523, 530, 534, 537, 543, 549, 555, 556, 558, 562, 566, 569, 577, 586, 596, 599, 603, 607, 611, 620, 624, 630, 636, 642, 644, 648, 654, 657, 661, 677, 684, 686, 688, 691, 700, 734, 758, 774, 789, 808, 851, 864, 866, 920, 925, 931, 942

automaton with array of counters, **67**, 179, 184, 188, 192, 195, 201, 236, 250, 254, 258, 266, 274, 278, 365, 438, 498, 562, 566, 569, 630, 644, 648, 700, 758, 931

automaton with counters, **68**, 207, 210, 213, 215, 220, 232, 244, 248, 285, 290, 300, 316, 352, 356, 404, 407, 416, 424, 449, 494, 523, 530, 534, 555, 577, 620, 624, 642, 734, 789, 808, 942

automaton without counters, **68**, 225, 230, 411, 453, 459, 462, 465, 469, 475, 477, 506, 537, 543, 549, 556, 558, 586, 596, 599, 603, 607, 611, 636, 654, 657, 661, 677, 684, 686, 688, 691, 774, 851, 864, 866, 920, 925

balance, 39, 64, 65, 67, 69, 83, 108, **250**, 252, 254, 256, 258, 260, 261, 907

balance_interval, 64, 65, 67, 69, 83, 88, 108, 250, **252**

balance_modulo, 64, 65, 67, 69, 83, 92, 108, 250, **256**

balance_partition, 64, 69, 83, 96, 108, 250, **260**, 543

balanced assignment, **69**, 250, 254, 258, 261

balanced tree, **69**, 907

Baptiste P., 444

Beldiceanu N., 206, 214, 222, 234, 238, 250, 264, 284, 302, 332, 342, 362, 378, 386, 418, 426, 436, 446, 512, 552, 554, 576, 584, 588, 592, 598, 606, 610, 624, 626, 632, 634, 640, 644, 646, 650, 652, 660, 670, 674, 676, 682, 696, 698, 722, 736, 738, 740, 752, 754, 760, 778, 782, 784, 786, 792, 794, 806, 876, 882, 892, 902, 930

Berge C., 176

Berge-acyclic constraint network, **69**, 506, 556, 558, 586, 596, 599, 603, 607, 611, 920, 925

Berliner H.J., 306

Bessière C., 698, 810, 814

bin_packing, 64, 65, 67, 99, 112, **264**, 365, 373, 553, 875

binary constraint, **70**, 465, 469, 477, 490, 851, 880

- binary_tree, 75, 86, 95, 107, **268**, 903
- bioinformatics, **70**, 173, 711, 774
- bipartite, **70**, 195, 215, 230, 274, 278, 281, 333, 337, 339, 341, 348, 356
- bipartite matching, **71**, 179, 180, 438, 582
- Bleuzen-Guernalec N., 176, 842
- Bockmayr A., 306, 726, 868, 896
- boolean channel, **71**, 453
- border, **71**, 736
- bound-consistency, **72**, 179, 498, 758, 931
- Bourdais S., 496, 538, 730
- Bourreau E., 386, 390, 398

- Cambazard H., 880
- card_matrix, **322**
- card_set, **24**
- card_var_gcc, **496**, 497
- cardinality_atleast, 62, 64, 65, 67, 70, 93, 108, **272**
- cardinality_atmost, 62, 64, 65, 67, 70, 93, 108, **276**
- cardinality_atmost_partition, 62, 65, 70, 93, 96, 108, **280**, 537
- cardinality_matrix, **322**, 323
- cardinality_on_attributes_values, **698**
- Carillon J.-P., 460
- Carlier J., 444
- Carlsson M., 222, 284, 302, 378, 426, 436, 584, 588, 592, 598, 606, 610, 670, 674, 680, 682, 698, 752, 792, 806, 876
- Carravilla M.A., 426
- case, 58, 462
- Caseau Y., 362
- CC, **47**, 778, 786
- centered cyclic(1) constraint network(1), **72**, 453, 537, 636, 654, 657, 691
- centered cyclic(2) constraint network(1), **73**, 459, 462, 465, 469, 477, 543, 661, 851

- centered cyclic(3) constraint network(1), **73**, 475, 677
- CHAIN*, **27**, 524
- change, 6, 25, 62, 66, 68, 93, 94, 101, 102, 106, 113, **284**, 298, 300, 302, 304, 314, 316, 402, 404, 407, 448, 449, 618, 620, 806, 808
- change_continuity, 25, 60, 62, 64, 66, 68, 75, 93, 97, 100–102, 106, 112, 114, **288**, 523, 525
- change_pair, 62, 66, 68, 93, 94, 96, 102, 106, 113, 285, **298**
- change_partition, 62, 93, 94, 96, 106, 285, **302**, 543
- channel routing, **74**, 343
- channeling constraint, **74**, 453, 569, 573, 615, 758
- choquet, 58, 112
- CIRCUIT*, **27**, 314, 854
- circuit, **74**, 307, 383, 387, 883
- circuit, 34, 36, 74, 86, 87, 89, 95, 97, **306**, 387, 868, 897
- circuit_cluster, 75, 86, 95, 178, **310**, 704
- circular sliding cyclic(1) constraint network(2), **74**, 316
- circular_change, 27, 66, 68, 74, 79, 94, 106, 285, **314**
- CLIQUE*, **27**, 176, 180, 182, 186, 190, 198, 250, 252, 256, 260, 268, 306, 310, 382, 386, 390, 394, 399, 418, 508, 568, 578, 622, 624, 626, 628, 632, 634, 638, 640, 644, 646, 650, 652, 656, 664, 666, 670, 674, 682, 688, 694, 696, 698, 704, 706, 708, 726, 740, 778, 794, 802, 814, 868, 892, 902, 906, 910, 911
- CLIQUE*($<$), 174, 358, 430, 432, 444, 512, 582, 712, 772, 810, 914
- CLIQUE*(\neq), 172, 318, 426, 446, 722, 744, 882, 896
- clique, 76, 86, 92, 112, **318**, 546, 615
- CLIQUE*(Comparison), **28**
- cluster, **75**, 312
- Colmerauer A., 176, 842
- colored_matrix, 91, 98, 106, **322**, 498, 758
- coloured, **75**, 236, 326, 330, 392, 562
- coloured_cumulative, 75, 94, 99, 100, 106, 112, **324**, 330, 365, 442, 704
- coloured_cumulatives, 75, 94, 99, 100, 106, 112, 326, **328**, 365, 704
- common, 62, 70, 76, 93, 194, 195, 207, **332**, 336–341, 438, 702, 764

- `common_interval`, 62, 70, 76, 88, 93, **336**
- `common_modulo`, 62, 70, 76, 92, 93, **338**
- `common_partition`, 62, 70, 76, 93, 96, **340**, 543
- conditional constraint, **75**, 782, 784
- Condon A.E., 172, 422
- `connect_points`, 28, 74, 85, 88, 104, 105, **342**
- connected component, **75**, 195, 270, 290, 312, 387, 392, 400, 506, 523, 578, 623, 702, 894, 903, 907, 911
- consecutive loops are connected, **76**, 523
- consecutive values, **76**, 632, 650, 696
- `constant_sum`, **874**, 874
- constraint between three collections of variables, **76**, 348, 848
- constraint between two collections of variables, **76**, 333, 337, 339, 341, 758, 762, 764, 767, 769, 771, 819, 821, 823, 825, 829, 833, 837, 839, 844, 931, 935, 937, 939
- constraint involving set variables, **76**, 180, 319, 490, 546, 573, 578, 615, 727, 776, 868, 880, 888, 891, 897
- constraint on the intersection, **77**, 195, 702, 764
- contact, **77**, 724, 920
- Contejean E., 206, 214, 222, 386, 418, 426, 882, 892
- convex, **77**, 506
- convex hull relaxation, **78**, 871
- Cormen T.H., 358
- Corn R.M., 172, 422
- correspondence, 62, 70, 76, 81, 93, 97, 178, **346**, 758, 848
- cost filtering constraint, **78**, 504, 668, 877, 948
- cost matrix, **78**, 504, 668
- `cost_gcc`, **502**
- Costa M.-C., 176
- count, 63, 66, 68, 79, 108, 207, 225, 236, **350**, 354, 356, 498, 562, 630, 646, 704
- counting constraint, **79**, 207, 210, 213, 215, 220, 352, 356, 435, 494, 671, 674, 682, 694, 700, 702, 704, 706

- counts, 62, 63, 66, 68, 70, 79, 93, 108, 235, 236, 352, **354**
- Cousin X., 560
- crossing, 24, 85, 89, 93, **358**, 514, 916
- cumulative, 5, 8, 9, 11, 25, 47–50, 64, 66, 67, 89, 98–100, 104, 106, 112, 232, 248, 264, 266, 324–326, 330, **362**, 366, 368, 370, 373, 375, 377, 381, 442, 444, 564, 875
- cumulative_product, 98–100, 106, 365, **366**, 748
- cumulative_trapeze, 58
- cumulative_two_d, 81, 85, 112, 365, **370**, 875
- cumulative_with_level_of_priority, 81, 99, 100, 106, 365, **374**, 875
- cumulatives, 7, 21, 80, 81, 98–100, 106, 110, 112, 328, 330, 365, **378**, 875
- cutset, 38, 74, 81, 86, 112, **382**, 546
- cycle, **79**, 387, 883
- cycle, 7, 38, 58, 74, 75, 79, 84, 86, 95, 97, 104, 179, **306**, 307, **386**, 392, 399, 400, 418, 419, 514, 569, 623, 883, 894, **896**, 897, 903
- cycle_card_on_path, 75, 86, 95, 100, 103, 215, 387, **390**
- cycle_change, 407
- cycle_or_accessibility, 84–86, 104, **394**, 706
- cycle_resource, 75, 81, 86, 99, 104, 113, 387, **398**
- cyclic, **79**, 316, 404, 407, 856
- cyclic_change, 62, 66, 68, 79, 93, 94, 101, 106, **402**, 406
- cyclic_change_joker, 62, 66, 68, 79, 88, 93, 94, 101, 106, **406**
- data constraint, **79**, 459, 462, 465, 469, 475, 477, 480, 484, 487, 539, 577, 677, 681, 851, 871
- decomposition, **80**, 173, 174, 223, 225, 230, 232, 411, 428, 430, 432, 444, 453, 549, 582, 589, 593, 615, 717, 774, 791, 793, 864, 866, 888, 891
- decomposition-based violation measure, **80**, 811
- decreasing, 66, 68, 80, 95, 101, **410**, 549, 864, 866
- deepest_valley, 66, 68, 92, 100, 101, **414**, 534
- demand profile, **80**, 381, 762
- derangement, 86, 97, 387, **418**
- derived collection, **81**, 236, 348, 373, 377, 381, 400, 453, 462, 475, 477, 487, 510, 537, 539, 543, 599, 603, 607, 611, 615, 661, 677, 681, 691, 799, 848, 901, 911, 916

Di Battista G., 914

`differ_from_at_least_k_pos`, 29, 63, 66, 68, 108, 109, 173, **422**

difference, **81**, 510

`diffn`, 5, 9, 26, 80, 85, 94, 96, 98, 104, 105, 161, 174, 370, 371, 373, **426**, 430, 432, 444, 716, 717, 741, 922, 924, 925, 929

`diffn_column`, 80, 85, 86, 96, 98, 428, **430**, 432, 922

`diffn_include`, 80, 85, 96, 98, 428, 430, **432**, 928

directed acyclic graph, **81**, 383

discrepancy, 79, 87, 89, 108, **434**, 546

disequality, **81**, 173, 179, 180, 438, 484, 510, 596, 688, 691, 811, 815, 883

disjoint, 36, 66, 67, 71, 81, 83, 108, **436**, 440, 442

`disjoint_tasks`, 94, 100, 106, 113, 438, **440**

disjunctive, 80, 99, 100, **444**

DISTANCE, **42**, 446, 448

`distance_between`, 98, **446**, 449

`distance_change`, 66, 68, 98, 102, 447, **448**

distinct, **7**

distribute, **496**

distribution, **496**, 497

domain channel, **82**, 453

domain definition, **82**, 225, 537, 691

`domain_constraint`, 20, 66, 68, 71, 72, 74, 80–82, 89, **452**, 614, 615, 887, 891

domination, **82**, 700, 877

`double_lex`, **580**

dual model, **83**, 569, 573

duplicated variables, **83**, 498, 599, 603, 607, 611

dynamic graph constraint

- assign_and_counts, 234
- assign_and_nvalues, 238
- bin_packing, 264
- circuit_cluster, 310
- coloured_cumulative, 324
- coloured_cumulatives, 328
- cumulative, 362
- cumulative_product, 366
- cumulative_two_d, 371
- cumulative_with_level_of_priority, 374
- cumulatives, 378
- cycle_card_on_path, 390
- cycle_or_accessibility, 394
- indexed_sum, 552
- interval_and_count, 560
- interval_and_sum, 564
- minimum_greater_than, 660
- next_element, 676
- next_greater_element, 680
- shift, 778
- sliding_card_skip0, 786
- sliding_time_window, 794
- sliding_time_window_sum, 802
- track, 900

Eades P., 914

egcc, **496**, 497

elem, 64, 66, 68, 73, 79, 85, 106, 109, **456**, 462, 851

element, 17–19, 51, 64, 66, 68, 73, 79, 81, 85, 101, 104, 106, 109, 113, 162, **456**, 456, 459, **460**, 464, 465, 468, 469, 473, **476**, 477, 480, 483, 484, 487, 539, 850, 851, 871

element_greatereq, 64, 66, 68, 70, 73, 79, 89, 106, 109, 459, 462, **464**, 469

element_lesseq, 64, 66, 68, 70, 73, 79, 89, 106, 109, 459, 462, 465, **468**

element_matrix, 8, 64, 66, 68, 73, 79, 81, 91, 106, 459, 462, **472**

element_sparse, 64, 66, 68, 70, 73, 79, 81, 104, 106, 109, 459, 462, **476**, 486, 487

- elements, 79, 85, 101, 106, 459, 462, **480**, 482
- elements_alldiff, **482**
- elements_alldifferent, 79, 81, 85, 97, 106, 459, 462, **482**
- elements_alldistinct, **482**
- elements_sparse, 79, 81, 101, 104, 106, **486**
- Elf M., 306, 726, 868, 896
- empty intersection, **83**, 438
- eq_set, 70, 76, 83, 98, **490**
- equality, **83**, 490
- equality between multisets, **83**, 758, 762
- equivalence, **83**, 250, 254, 258, 261, 630, 648, 671, 674, 682, 688, 694, 700, 704, 815
- Erschler J., 362
- Euler knight, **84**, 387
- exactly, 63, 66–68, 79, 108, 207, 244, 248, **492**
- excluded, **84**, 691
- extension, **84**, 539
- extension, **538**

- facilities location problem, **84**, 396, 877
- Fages F., 382
- Fahle T., 318
- Falkenhainer B., 784
- Flajolet O., 622
- Flamm C., 772
- Flener P., 580, 862, 902
- flow, **84**, 498, 501, 758, 811, 888, 891, 931
- Focacci F., 434, 666
- frequency allocation problem, **85**, 174
- Frisch A., 204, 580, 598, 606, 610, 862
- Frutos A.G., 172, 422

functional dependency, **85**, 459, 462, 480, 484, 851

Galinier P., 496, 538, 730

Gambini I., 426

Garey M.R., 914

gcc, **496**

gccc, **502**

Gent I., 708

geometrical constraint, **85**, 343, 360, 373, 396, 428, 430, 432, 514, 713, 719, 721, 724, 742, 746, 774, 916, 920, 922, 925, 929

Ginsberg L., 434

global_cardinality, 46, 64–67, 72, 83, 84, 87, 90, 108, 113, 207, 236, 272, 274, 276, 278, 280, 281, 322, 323, **496**, 501, 504, 630, 646, 756, 758, 760, 762, 789, 886, 888, 890, 891

global_cardinality_low_up, 64, 65, 84, 108, **500**, 791

global_cardinality_with_costs, 23, 42, 64, 78, 90, 100, 110, 498, **502**, 668, 877, 948

global_contiguity, 28, 44, 46, 51–53, 66–69, 75, 77, **506**

golomb, 22, 81, 85, **508**

Golomb ruler, **85**, 510

Golomb S.W., 508, 744

Golynski A., 496

Gomes C., 322

graph constraint, **86**, 270, 307, 312, 319, 383, 387, 392, 396, 400, 419, 569, 578, 623, 711, 727, 868, 883, 894, 897, 903, 907, 911

Graph invariants:

MAX_NCC, 121

MAX_NSCC, 121

MIN_NCC, 121

MIN_NSCC, 121

NARC, 121

NCC, 122

NSCC, 122

NSINK, 122

NSOURCE, 122

NVERTEX, 122
MAX_NCC, MAX_NSCC, 123
MAX_NCC, MIN_NCC, 123
MAX_NCC, NARC, 123
MAX_NCC₁, NCC₂, 150
MAX_NCC₂, NCC₁, 150
MAX_NCC, NSINK, 124
MAX_NCC, NSOURCE, 124
MAX_NCC, NVERTEX, 124
MAX_NSCC, MIN_NSCC, 125
MAX_NSCC, NARC, 125
MAX_NSCC, NVERTEX, 125
MIN_NCC, MIN_NSCC, 125
MIN_NCC, NARC, 126
MIN_NCC, NCC, 126
MIN_NCC₁, NCC₂, 150
MIN_NCC₂, NCC₁, 151
MIN_NCC, NVERTEX, 126
MIN_NSCC, NARC, 127
MIN_NSCC, NVERTEX, 127
NARC₁, NARC₂, 151
NARC, NCC, 127
NARC, NSCC, 127
NARC, NVERTEX, 128
NCC₁, NCC₂, 151
NCC, NSCC, 129
NCC, NVERTEX, 129
NSCC, NVERTEX, 130
NSINK, NVERTEX, 130
NSOURCE, NVERTEX, 130
NVERTEX₁, NVERTEX₂, 151
MAX_NCC₁, MIN_NCC₁, MIN_NCC₂, 152
MAX_NCC₂, MIN_NCC₂, MIN_NCC₁, 152
MAX_NCC, MIN_NCC, NARC, 131
MAX_NCC, MIN_NCC, NCC, 131
MAX_NCC, MIN_NCC, NVERTEX, 131
MAX_NCC, NARC, NCC, 132

- MAX_NCC, NARC, NVERTEX, 133
 MAX_NCC, NCC, NVERTEX, 134
 MAX_NSCC, MIN_NSCC, NARC, 134
 MAX_NSCC, MIN_NSCC, NSCC, 135
 MAX_NSCC, MIN_NSCC, NVERTEX, 135
 MAX_NSCC, NSCC, NVERTEX, 136
 MIN_NCC₁, NARC₂, NCC₁, 153
 MIN_NCC, NARC, NVERTEX, 136
 MIN_NCC, NCC, NVERTEX, 137
 MIN_NSCC, NARC, NVERTEX, 138
 MIN_NSCC, NSCC, NVERTEX, 138
 NARC, NCC, NVERTEX, 138
 NARC, NSCC, NVERTEX, 140
 NARC, NSINK, NVERTEX, 142
 NARC, NSOURCE, NVERTEX, 143
 NSINK, NSOURCE, NVERTEX, 143
 MAX_NCC₁, MIN_NCC₁, MIN_NCC₂, NCC₁, 153
 MAX_NCC₂, MIN_NCC₂, MIN_NCC₁, NCC₂, 154
 MAX_NCC, MIN_NCC, NARC, NCC, 144
 MAX_NCC, MIN_NCC, NCC, NVERTEX, 145
 MAX_NSCC, MIN_NSCC, NARC, NSCC, 145
 MAX_NSCC, MIN_NSCC, NSCC, NVERTEX, 145
 MIN_NCC, NARC, NCC, NVERTEX, 146
 NARC, NCC, NSCC, NVERTEX, 147
 NARC, NSINK, NSOURCE, NVERTEX, 149
 MAX_NCC₁, MAX_NCC₂, MIN_NCC₁, MIN_NCC₂, NCC₁, 154
 MAX_NCC₁, MAX_NCC₂, MIN_NCC₁, MIN_NCC₂, NCC₂, 157
 MAX_NCC, MIN_NCC, NARC, NCC, NVERTEX, 149
 MIN_NCC, NARC, NCC, NSCC, NVERTEX, 149
 MAX_NCC₁, MAX_NCC₂, MIN_NCC₁, MIN_NCC₂, NCC₁, NCC₂, 159
- graph partitioning constraint, 86, 270, 307, 387, 400, 623, 883, 894, 903, 907, 911
 graph_crossing, 85, 89, 360, 512, 623, 903, 916
 GRID, 28
 GRID([SIZE1, SIZE2, SIZE3]), 342
 Grinberg E.Ya., 306

group, 35, 63, 66, 68, 75, 76, 106, 110, 114, 118, 290, 506, **516**, 524, 525, 537, 691, 730, 856, 860

group_skip_isolated_item, 27, 63, 66, 68, 104, 106, 290, 523, **524**, 537

guillotine cut, **86**, 430, 922

Guo Q., 426

Hall interval, **87**, 179, 498

Hamiltonian, **87**, 307, 897

Harary F., 914

Harvey W.D., 434

Hebrard E., 698

heighest_peak, 66, 68, 100, 101, 416, **532**

Hellsten L., 854, 858

Henz M., 882

heuristics, **87**, 435

Hnich B., 580, 598, 606, 610, 698, 862

Hofacker I.L., 772

Hooker J.N., 362, 464, 468

hypergraph, **87**, 223, 232, 713, 753, 782, 784, 791, 793

in, 29, 66, 68, 72, 81, 82, 87, 107, 108, 162, 281, **536**, 543, 690, 691

in_attr, **6**

in_list, **6**

in_relation, 10, 19, 79, 81, 84, 99, 107, **538**, 944, 945

in_same_partition, 66, 68, 73, 81, 96, 108, 198, 304, 341, 537, **542**, 671, 771, 939

in_set, 76, 87, 98, 108, **546**

included, **87**, 537, 546

inclusion, **88**, 931, 935, 937, 939

increasing, 66, 68, 80, 95, 101, 410, 411, **548**, 864, 866

increasing_seq, **7**

indexed_sum, 64, 109, **552**, 875

- indistinguishable values, **88**, 556, 558, 776
- inequality_sum, 58
- inflexion, 51, 53, 66, 68, 100, 101, 506, **554**, 732, 734, 940, 942
- int_value_precede, 66, 68, 69, 88, 95, 105, 109, **556**, 558, 776
- int_value_precede_chain, 66, 68, 69, 88, 95, 105, 109, 556, **558**
- interval, **88**, 188, 213, 254, 337, 562, 566, 682, 767, 819, 829, 935
- interval_and_count, 64–67, 75, 88, 99, 106, 112, 215, **560**
- interval_and_sum, 64–67, 88, 99, 106, 112, **564**, 875
- inverse, 59, 66, 67, 74, 83, 86, 93, 97, 387, **568**, 572, 573
- inverse_set, 12, 74, 76, 83, 100, 546, 569, **572**
- ith_pos_different_from_0, 63, 66, 68, 79, 88, 106, **576**, 644

- Jünger M., 306, 726, 868, 896
- Jackson, 712
- Jefferson C., 204
- Johnson D.S., 914
- joker value, **88**, 184, 210, 343, 407, 577, 657, 706, 738, 948
- Jussien N., 854

- k – diff, 700
- k_cut, 75, 76, 86, 89, 546, **578**
- Kasper T., 306, 726, 868, 896
- Katriel I., 496, 754, 760, 930
- Kocjan W., 886, 890
- Kreuger P., 886, 890
- Kuchcinski K., 426
- Kızıltan Z., 580, 598, 606, 610, 698, 754, 862

- López-Ortiz A., 176, 496
- Labbé M., 394, 892
- Laburthe F., 362
- Lahrichi A., 362, 370

- Lal A., 382
- Laporte G., 310, 394, 892
- Laurière J.-L., 176, 306, 386
- Law Y.C., 556, 558, 776
- Le Pape C., 444
- Leconte M., 176
- Lee J.H.M., 556, 558, 776
- Leiserson C.E., 358
- Levy H., 382
- `lex2`, 89, 91, 95, 98, 105, 204, **580**, 862
- `lex_alldiff`, **582**
- `lex_alldifferent`, 71, 80, 109, 179, **582**, 596
- `lex_alldistinct`, **582**
- `lex_between`, 66, 68, 69, 89, 95, 105, 109, **584**, 589, 593, 599, 603, 607, 611
- `lex_chain`, **588**, **592**
- `lex_chain_less`, 80, 89, 91, 95, 105, 109, 584, 586, **588**, 593, 599, 603, 607, 611
- `lex_chain_lesseq`, 80, 89, 91, 95, 105, 109, 580, 584, 586, 589, **592**, 599, 603, 607, 611, 862
- `lex_different`, 66, 68, 69, 81, 109, 582, **596**
- `lex_greater`, 66, 68, 69, 81, 83, 89, 91, 93, 95, 105, 109, 586, 589, 593, **598**, 603, 607, 611
- `lex_greatereq`, 66, 68, 69, 81, 83, 89, 91, 93, 95, 105, 109, 586, 589, 593, 596, 599, **602**, 607, 611
- `lex_less`, 66, 68, 69, 81, 83, 89, 91, 93, 95, 105, 109, 586, 589, 593, 596, 599, 603, **606**, 611
- `lex_lesseq`, 19, 20, 29, 40, 51–53, 66, 68, 69, 81, 83, 89, 91, 93, 95, 105, 109, 204, 580, 586, 589, 593, 596, 599, 603, 607, **610**, 862
- lexicographic order, **89**, 204, 580, 586, 589, 593, 599, 603, 607, 611, 862
- limited discrepancy search, **89**, 435
- line-segments intersection, **89**, 360, 514, 916
- linear programming, **89**, 307, 365, 453, 465, 469, 578, 615, 727, 868, 871, 897
- `link_set_to_booleans`, 74, 76, 80, 81, 89, 100, 108, 180, 319, 453, 546, 578, **614**, 727, 868, 887, 888, 891, 897
- Liu Q., 172, 422

Lloyd E.L., 382

Lodi A., 666

longest_change, 34, 66, 68, 102, 106, 285, **618**

LOOP, **28**, 506, 516, 517, 786, 854, 858

Lopez P., 362

Lorca X., 902

Low D.W., 382

Lubiw A., 580

Müller T., 882

Müller-Hannemann M., 264

magic hexagon, **90**, 504

magic series, **90**, 498

magic square, **90**, 504

Maher M., 506

map, 75, 86, 387, 514, **622**, 903

Marte M., 900

Martello S., 264

matching, **91**, 883

matrix, **91**, 204, 323, 475, 580, 862

matrix model, **91**, 204, 323, 580, 862

matrix symmetry, **91**, 580, 589, 593, 599, 603, 607, 611

MAX_DRG, **34**

MAX_ID, **34**, 268, 272, 276, 280, 306, 892, 896, 946

max_index, 64, 66, 68, 91, 95, **624**, 642

max_n, 91, 95, 99, **626**, 644

MAX_NCC, **34**, 194, 289, 290, 517, 618, 854, 858

MAX_NSCC, **35**, 176, 180, 182, 186, 190, 198, 200, 268, 382, 508, 525, 628, 632, 708, 902, 906, 910

max_nvalue, 64, 66, 67, 83, 91, 92, 108, 207, 352, 498, **628**, 648

MAX_OD, **35**, 896

`max_size`, **10**

`max_size_set_of_consecutive_var`, 76, 91, 108, **632**

`maximum`, **91**, 624, 626, 630, 632, 636, 638

`maximum`, 66, 68, 72, 91, 95, 626, **634**, 638, 654

`maximum clique`, **92**, 319

`maximum number of occurrences`, **92**, 630

`maximum_modulo`, 91, 92, 95, **638**, 664

`maxint`, **92**, 416, 644, 654, 657, 664

Mehlhorn K., 176, 306, 726, 842, 868, 896

Miguel I., 204, 580, 598, 606, 610, 862

Milano M., 666

MIN_DRG, **35**

MIN_ID, **35**, 896

`min_index`, 64, 66, 68, 92, 95, 624, **640**

`min_n`, 66, 67, 92, 95, 99, 576, 626, **644**

MIN_NCC, **35**, 289, 290, 517, 854, 858

MIN_NSCC, **36**, 306, 525, 646, 650, 868, 896

`min_nvalue`, 64, 66, 67, 83, 92, 108, 207, 352, 498, 630, **646**, 657

MIN_OD, **36**, 896

`min_size`, **10**

`min_size_set_consecutive_var`, 65

`min_size_set_of_consecutive_var`, 76, 92, 108, **650**, 696

`min_weight_alldiff`, **666**

`min_weight_alldifferent`, **666**

`min_weight_alldistinct`, **666**

`minimum`, **92**, 642, 644, 648, 650, 654, 657, 661, 664, 677, 681

`minimum`, 26, 39, 51, 66, 68, 72, 92, 95, 634, 636, 644, **652**, 656, 657, 664

`minimum number of occurrences`, **92**, 648

`minimum_distance`, **174**

- minimum_except_0, 66, 68, 72, 88, 92, 95, **656**
- minimum_greater_than, 66, 68, 73, 81, 92, 95, 654, **660**, 677, 681
- minimum_modulo, 92, 95, 638, **664**
- minimum_weight_alldiff, **666**
- minimum_weight_alldifferent, 65, 78, 95, 110, **666**, 877, 948
- minimum_weight_alldistinct, **666**
- Mittal S., 784
- modulo, **92**, 192, 220, 258, 339, 638, 664, 769, 821, 833, 937
- multiset, **93**, 758, 762
- multiset ordering, **93**, 599, 603, 607, 611

- n-queen, **93**, 179, 569

- NARC**, **36**, 172, 174, 206, 208, 212, 214, 218, 222, 224, 228, 232, 242, 246, 284, 289, 290, 298, 302, 314, 318, 324, 328, 346, 350, 354, 358, 362, 366, 371, 374, 378, 402, 406, 410, 422, 426, 430, 432, 434, 436, 440, 444, 452, 456, 460, 464, 468, 472, 476, 480, 492, 512, 536, 538, 548, 568, 572, 582, 588, 592, 596, 614, 660, 676, 680, 690, 712, 716, 718, 720–722, 740, 752, 772, 778, 782, 784, 790, 792, 802, 806, 810, 842, 846, 850, 864, 866, 882, 886, 890, 896, 900, 914, 918, 922, 924, 928, 944

- NARC_NO_LOOP**, **36**, 200

- nbchanges, **284**

- NCC**, **37**, 268, 289, 290, 386, 390, 394, 399, 506, 517, 578, 622, 702, 708, 722, 745, 892, 902, 906, 910

- nclass, 79, 83, 94, 96, 104, 108, 543, **670**, 674, 682, 694

- nequivalence, 79, 83, 94, 104, 108, 671, **674**, 682, 694

- next_element, 66, 68, 73, 79, 81, 92, 106, 654, **676**, 681

- next_greater_element, 79, 81, 92, 95, 106, 654, 660, 661, 677, **680**

- ninterval, 12, 79, 83, 88, 94, 104, 108, 671, 674, **682**, 694

- no_cycle, 58

- no_loop, **93**, 173, 195, 215, 230, 274, 278, 281, 285, 290, 300, 304, 333, 337, 339, 341, 348, 356, 360, 404, 407

- no_peak, 66, 68, 100, 101, **684**, 686, 734

- no_valley, 66, 68, 100, 101, 684, **686**, 942

- non-overlapping, **94**, 428, 442, 721, 724, 742, 920, 925

- `not_all_equal`, 66, 68, 81, 83, 101, 108, **688**, 700
- `not_in`, 66, 68, 72, 81, 82, 84, 107, 108, 537, **690**
- `npair`, 79, 83, 94, 96, 104, 108, 671, 674, 682, **694**, 700
- `NSCC`, **37**, 310, 342, 525, 670, 674, 682, 688, 694, 696, 698, 704, 706, 814
- `nset_of_consecutive_values`, 76, 104, 108, 632, 650, **696**
- `NSINK`, **37**, 332, 336, 338, 340, 542, 754, 760, 764, 766, 768, 770, 842, 930, 934, 936, 938
- `NSINK_NSOURCE`, **37**, 818, 820, 822, 824, 828, 832, 836, 838
- `NSOURCE`, **38**, 332, 336, 338, 340, 486, 542, 754, 760, 764, 766, 768, 770, 842, 876, 930, 934, 936, 938
- `NTREE`, **38**, 310, 386, 390, 394, 399, 418, 622, 666
- number of changes, **94**, 285, 300, 304, 316, 404, 407, 808
- number of distinct equivalence classes, **94**, 671, 674, 682, 694, 700, 704
- number of distinct values, **94**, 240, 326, 330, 700, 702, 704, 706
- `nvalue`, 15–17, 37, 45, 66, 67, 79, 82, 84, 94, 104, 108, 207, 208, 210, 240, 352, 498, 628, 630, 648, 670, 671, 674, 682, 688, 694, **698**, 702, 704, 706, 877, 901
- `nvalue_on_intersection`, 75, 77, 79, 94, 195, 333, 700, **702**, 764
- `nvalues`, 79, 84, 94, 104, 108, 238, 240, 312, 324, 326, 328, 330, 700, **704**, 706
- `nvalues_except_0`, 79, 88, 94, 104, 108, 396, 700, 704, **706**
- `NVERTEX`, **38**, 318, 382, 395, 399, 482, 496, 500, 502, 517, 525, 708, 722, 745, 760, 892, 910, 911
- obscure, **94**, 711
- Older W.J., 754, 842
- one succ, **95**
- `one_factor`, **882**, 883
- `one_machine`, **444**
- `one_succ`, 179, 180, 184, 188, 192, 198, 270, 307, 312, 387, 392, 668, 903
- `one_tree`, 6, 70, 86, 94, 97, 107, **708**
- orchard, 28, 63, 85, 87, **712**
- `ORDER`, **39**, 624, 626, 634, 638, 640, 644, 652, 656, 664
- order constraint, **95**, 204, 411, 549, 556, 558, 580, 586, 589, 593, 599, 603, 607, 611, 624, 626, 636, 638, 642, 644, 654, 657, 661, 664, 681, 776, 862, 864, 866

- orth_link_ori_siz_end, 80, 96, 428, **716**
- orth_on_the_ground, 85, 96, 717, **718**, 741
- orth_on_to_of_orth, 85
- orth_on_top_of_orth, 94, 96, 717, **720**, 741
- orthotope, **96**, 428, 430, 432, 717, 719, 721, 724, 742, 920, 922, 925, 929
- orths_are_connected, 77, 85, 94, 96, 107, 716, 717, **722**, 918, 920
- Ottosson G., 464, 468
- Péridy L., 444
- Pachet F., 284, 698
- pair, **96**, 300, 694
- partition, **96**, 198, 261, 281, 304, 341, 543, 671, 771, 823, 837, 939
- PATH*, **28**, 222, 284, 289, 298, 302, 402, 406, 410, 448, 506, 516, 517, 548, 588, 592, 618, 680, 752, 786, 790, 792, 806, 842, 846, 850, 858, 864, 866
- path, **96**, 727, 894
- path, **726**, 894
- PATH_1*, **28**, 232, 784
- PATH.FROM.TO**, **39**, 598, 602, 606, 610, 726
- path_from_to, 76, 86, 89, 96, 546, 615, **726**, 894
- PATH.LENGTH**, **48**
- PATH.LENGTH(PATH_LEN), 390
- PATH_N*, **29**, 782
- pattern, 51, 98, 103, 106, 285, **730**, 791, 856, 860
- peak, 66, 68, 100, 101, 532, 534, 555, 684, 686, **732**, 942
- Pearson J., 580, 582, 772, 862
- pentomino, **96**, 746
- period, 71, 97, 98, 100, 106, **736**, 738
- period_except_0, 88, 97, 98, 100, 106, 736, **738**
- periodic, **97**, 736, 738
- permutation, **97**, 179, 290, 307, 348, 387, 419, 484, 569, 758, 762, 767, 769, 771, 844, 848, 883
- permutation channel, **97**, 569

- Pesant G., 496, 538, 730, 854, 858, 882
- Petit T., 810, 814
- phylogeny, **97**, 711
- pick-up delivery, **97**, 387
- place_in_pyramid, 59, 85, 94, 96, 428, 716, 718–721, **740**
- Poder E., 736, 738
- polygon, **98**, 428
- polyomino, 85, 96, 104, **744**
- positioning constraint, **98**, 430, 432, 922, 929
- PRED, **48**, 395
- predefined constraint, **98**, 204, 323, 490, 546, 580, 731, 736, 738, 862
- producer-consumer, **98**, 365, 381
- PRODUCT*, **29**, 194, 214, 234, 238, 264, 272, 276, 280, 324, 328, 332, 336, 338, 340, 346, 354, 362, 366, 371, 375, 378, 436, 440, 452, 456, 460, 464, 468, 472, 476, 480, 482, 486, 502, 536, 538, 542, 552, 560, 564, 572, 614, 660, 676, 680, 690, 702, 754, 760, 764, 766, 768, 770, 798, 818, 820, 822, 824, 828, 832, 836, 838, 842, 846, 850, 870, 876, 886, 890, 900, 930, 934, 936, 938, 946
- PRODUCT*(=), 228, 422, 596, 720, 918, 922, 928, 944
- PRODUCT*(*CLIQUE*, *LOOP*, =), 200
- PRODUCT*(*PATH*, *VOID*), 598, 602, 606, 610
- PRODUCT**, **40**, 748
- product, **98**, 368, 748
- PRODUCT*(Comparison), **29**
- product_ctr, 40, 64, 98, **748**, 750, 875
- Prosser P., 708
- proximity constraint, **98**, 201, 447, 449
- Puget J.-F., 176, 790
- Quimper C.-G., 176, 496
- Régin J.-C., 174, 176, 272, 276, 318, 322, 496, 500, 502, 698, 790, 810, 814, 882
- RANGE**, **40**, 750

- range, **99**, 750
- range, 58
- range_ctr, 41, 64, 99, 748, **750**, 875
- RANGE_DRG**, **38**, 906
- RANGE_NCC**, **39**
- RANGE_NSCC**, **39**, 250, 252, 256, 260
- rank, **99**, 626, 644
- Refalo P., 452
- regular, 58
- relation, **99**, 539, 888, 891
- relaxation, **99**, 184, 753, 811, 815, 819, 821, 823, 825, 829, 833, 837, 839, 877, 948
- relaxed_sliding_sum, 12, 87, 99, 100, 103, **752**, 875
- require_at_least, **8**
- required, **8**
- resource constraint, **99**, 266, 326, 330, 365, 368, 377, 381, 400, 444, 562, 566, 901, 911
- rgcc, 497
- Ribeiro C., 426
- Rivest R.L., 358
- Rivreau D., 444
- Rochart G., 854
- Rodríguez-Martín I., 394, 892
- roots, 58
- Rousseau J.M., 496
- row_and_column_lex, **580**
- Roy P., 284, 698
- run of a permutation, **100**, 290
- Sadjad S.B., 496
- same, 29, 37, 38, 44, 66, 67, 72, 74, 76, 83, 84, 93, 97, 323, 347, 348, **754**, 760, 762, 764, 766–771, 825, 844, 931
- same_and_gcc, **760**

- same_and_global_cardinality, 65, 76, 80, 83, 93, 97, 108, 498, 756, 758, **760**
- same_gcc, **760**
- same_intersection, 76, 77, 195, 333, 702, 758, **764**
- same_interval, 76, 88, 97, 758, **766**, 818, 819
- same_modulo, 76, 92, 97, 758, **768**, 820, 821
- same_partition, 76, 96, 97, 543, 758, **770**, 822, 823
- same_size, **9**
- same_with_cardinalities, **760**
- Samet H., 370
- Sanner A.M.W., 172, 422
- scalar product, **100**, 504
- scheduling constraint, **100**, 326, 330, 365, 368, 377, 381, 442, 444, 736, 738, 780
- Schwenk A.J., 914
- Sedgewick R., 622
- SELF*, **29**, 206, 208, 212, 218, 224, 242, 246, 324, 328, 350, 362, 366, 370, 374, 378, 426, 434, 440, 492, 496, 500, 502, 716, 718, 722, 748, 750, 760, 772, 778, 802, 874, 880, 900
- Sellman M., 666
- sequence, **100**, 223, 232, 392, 416, 534, 555, 684, 686, 734, 736, 738, 753, 774, 782, 784, 789, 791, 793, 942
- sequence_folding, 66, 68, 70, 80, 85, 100, **772**
- set channel, **100**, 573, 615
- set_value_precede, 77, 88, 95, 98, 105, 109, 556, **776**
- sgcc, **760**, **890**
- shared table, **101**, 480, 487
- Shaw P., 264
- Shearer J.B., 508
- shift, 101, 106, 750, **778**, 796
- Shufet J.A., 306
- sign, **24**

signature

AUTOMATON

deepest_valley, 414
 heighest_peak, 532
 inflexion, 554
 int_value_precede, 556
 int_value_precede_chain, 558
 ith_pos_different_from_0, 576
 lex_between, 584
 no_peak, 684
 no_valley, 686
 peak, 732
 valley, 940

CC(NSINK, NSOURCE), PRODUCT

same_intersection, 764

CLIQUE, SUCC

sliding_time_window, 794

DISTANCE, CLIQUE(\neq)

distance_between, 446

DISTANCE, PATH

distance_change, 448

MAX_ID, MAX_NSCC, NCC, CLIQUE

binary_tree, 268

MAX_ID, MIN_NSCC, CLIQUE

circuit, 306

MAX_ID, NCC, NVERTEX, CLIQUE

temporal_path, 892

MAX_ID, PRODUCT

cardinality_atleast, 272
 cardinality_atmost, 276
 cardinality_atmost_partition, 280

MAX_ID, SUM, PRODUCT

weighted_partial_alldiff, 946

MAX_NCC, CIRCUIT, LOOP, \forall

stretch_circuit, 854

MAX_NCC, MIN_NCC, NARC, NCC, PATH

change_continuity, 288

- MAX_NCC, MIN_NCC, NCC, NVERTEX, *PATH*, *LOOP*; MAX_NCC, MIN_NCC, *PATH*, *LOOP*
 group, 516
- MAX_NCC, *PATH*
 longest_change, 618
- MAX_NCC, *PATH*, *LOOP*, \forall
 stretch_path, 858
- MAX_NCC, *PRODUCT*
 alldifferent_on_intersection, 194
- MAX_NSCC, *CLIQUE*
 alldifferent, 176
 alldifferent_between_sets, 180
 alldifferent_except_0, 182
 alldifferent_interval, 186
 alldifferent_modulo, 190
 alldifferent_partition, 198
 golomb, 508
- MAX_NSCC, *CLIQUE*
 max_nvalue, 628
 max_size_set_of_consecutive_var, 632
- MAX_NSCC, MIN_NSCC, NSCC, NVERTEX, *CHAIN*
 group_skip_isolated_item, 524
- MAX_NSCC, NARC_NO_LOOP, *PRODUCT*(*CLIQUE*, *LOOP*, =)
 alldifferent_same_value, 200
- MAX_NSCC, NCC, *CLIQUE*
 tree, 902
- MAX_NSCC, NCC, NVERTEX, *CLIQUE*
 one_tree, 708
- MAX_NSCC, NCC, NVERTEX, *CLIQUE*; NVERTEX, *CLIQUE*, \forall
 tree_resource, 910
- MAX_NSCC, NCC, RANGE_DRG, *CLIQUE*
 tree_range, 906
- MAX_NSCC, NVERTEX, *CLIQUE*
 cutset, 382
- MIN_NSCC, *CLIQUE*
 min_nvalue, 646
 min_size_set_of_consecutive_var, 650
 strongly_connected, 868

NARC, *CIRCUIT*

circular_change, 314

NARC, *CLIQUE*($<$)

all_min_dist, 174

diffn_column, 430

diffn_include, 432

disjunctive, 444

lex_alldifferent, 582

NARC, *CLIQUE*(\neq)

all_differ_from_at_least_k_pos, 172

NARC, *CLIQUE*

inverse, 568

place_in_pyramid, 740

NARC, *CLIQUE*($<$)

crossing, 358

graph_crossing, 512

orchard, 712

soft_alldifferent_ctr, 810

two_layer_edge_crossing, 914

NARC, *CLIQUE*(\neq)

symmetric_alldifferent, 882

NARC, *CLIQUE*(\neq); MAX_ID, MAX_OD, MIN_ID, MIN_NSCC, MIN_OD, *CLIQUE*(\neq)

tour, 896

NARC, NVERTEX, *CLIQUE*(\neq)

clique, 318

NARC, *PATH*

among_seq, 222

decreasing, 410

increasing, 548

lex_chain_less, 588

lex_chain_lesseq, 592

sliding_distribution, 790

sliding_sum, 792

strictly_decreasing, 864

strictly_increasing, 866

NARC, *PATH*

change, 284
 change_pair, 298
 change_partition, 302
 cyclic_change, 402
 cyclic_change_joker, 406
 relaxed_sliding_sum, 752
 smooth, 806
NARC, *PATH_1*
 arith_sliding, 232
 size_maximal_starting_sequence_alldifferent, 784
NARC, *PATH_N*
 size_maximal_sequence_alldifferent, 782
NARC, *PATH*; NARC, *PRODUCT*
 stage_element, 850
NARC, *PATH*; NARC, *PRODUCT*, *SUCC*
 next_greater_element, 680
NARC, *PRODUCT*
 element_sparse, 476
 in_relation, 538
NARC, *PRODUCT*(=)
 differ_from_at_least_k_pos, 422
 lex_different, 596
NARC, *PRODUCT*
 disjoint, 436
 not_in, 690
NARC, *PRODUCT*
 among_low_up, 214
 correspondence, 346
 counts, 354
 domain_constraint, 452
 elem, 456
 element, 460
 element_greatereq, 464
 element_lesseq, 468
 element_matrix, 472
 elements, 480
 in, 536

- inverse_set, 572
- link_set_to_booleans, 614
- symmetric_cardinality, 886
- symmetric_gcc, 890
- $\overline{\text{NARC}}$, *PRODUCT*(=)
 - arith_or, 228
 - orth_on_top_of_orth, 720
 - two_orth_are_in_contact, 918
 - two_orth_column, 922
 - two_orth_include, 928
 - vec_eq_tuple, 944
- $\overline{\text{NARC}}$, *PRODUCT*; $\overline{\text{NARC}}$, *PATH*
 - sort_permutation, 846
- $\overline{\text{NARC}}$, *PRODUCT*, *SUCC*
 - minimum_greater_than, 660
 - next_element, 676
- $\overline{\text{NARC}}$, *SELF*
 - arith, 224
 - atleast, 242
 - orth_link_ori_siz_end, 716
- $\overline{\text{NARC}}$, *SELF*
 - atmost, 246
- $\overline{\text{NARC}}$, *SELF*
 - among, 206
 - among_diff_0, 208
 - among_interval, 212
 - among_modulo, 218
 - count, 350
 - discrepancy, 434
 - exactly, 492
 - orth_on_the_ground, 718
- $\overline{\text{NARC}}$, *SELF*; *CLIQUE*, *CC*
 - shift, 778
- $\overline{\text{NARC}}$, *SELF*; *CLIQUE*, *SUCC*
 - sliding_time_window_sum, 802
- $\overline{\text{NARC}}$, *SELF*; $\overline{\text{NARC}}$, *CLIQUE*($<$)
 - sequence_folding, 772

$\overline{\text{NARC}}, \text{SELF}; \overline{\text{NARC}}, \text{CLIQUE}(\neq)$
 diffn, 426
 $\overline{\text{NARC}}, \text{SELF}; \underline{\text{NARC}}, \text{PRODUCT}$
 disjoint_tasks, 440
 $\overline{\text{NARC}}, \text{SELF}; \overline{\text{NCC}}, \overline{\text{NVERTEX}}, \text{CLIQUE}(\neq)$
 orths_are_connected, 722
 $\overline{\text{NARC}}, \text{SELF}; \text{PRODUCT}, \forall, \text{SUCC}$
 coloured_cumulatives, 328
 cumulative_with_level_of_priority, 374
 cumulatives, 378
 $\overline{\text{NARC}}, \text{SELF}; \text{PRODUCT}, \text{SUCC}$
 coloured_cumulative, 324
 cumulative, 362
 cumulative_product, 366
 cumulative_two_d, 370
 track, 900
 $\overline{\text{NARC}}, \text{SYMMETRIC_PRODUCT}(=)$
 two_orth_do_not_overlap, 924
 $\overline{\text{NCC}}, \text{CLIQUE}$
 k_cut, 578
 $\overline{\text{NCC}}, \underline{\text{NTREE}}, \text{CLIQUE}$
 cycle, 386
 $\overline{\text{NCC}}, \overline{\text{NTREE}}, \text{CLIQUE}$
 map, 622
 $\overline{\text{NCC}}, \underline{\text{NTREE}}, \text{CLIQUE}; \overline{\text{NVERTEX}}, \text{CLIQUE}, \text{PRED}$
 cycle_or_accessibility, 394
 $\overline{\text{NCC}}, \underline{\text{NTREE}}, \text{CLIQUE}, \text{PATH_LENGTH}$
 cycle_card_on_path, 390
 $\overline{\text{NCC}}, \underline{\text{NTREE}}, \overline{\text{NVERTEX}}, \text{CLIQUE}; \overline{\text{NVERTEX}}, \text{CLIQUE}, \forall$
 cycle_resource, 398
 $\overline{\text{NCC}}, \overline{\text{NVERTEX}}, \text{CLIQUE}(\neq)$
 polyomino, 744
 $\underline{\text{NCC}}, \text{PATH}, \text{LOOP}$
 global_contiguity, 506
 $\overline{\text{NCC}}, \text{PRODUCT}$
 nvalue_on_intersection, 702
 $\overline{\text{NSCC}}, \text{CLIQUE}$

- not_all_equal, 688
- NSCC, *CLIQUE*
- nclass, 670
- nequivalence, 674
- ninterval, 682
- npair, 694
- nset_of_consecutive_values, 696
- nvalue, 698
- nvalues, 704
- nvalues_except_0, 706
- soft_alldifferent_var, 814
- NSCC, *GRID*([SIZE1, SIZE2, SIZE3])
 - connect_points, 342
- NSCC, NTREE, *CLIQUE*, ALL_VERTICES
 - circuit_cluster, 310
- NSINK_NSOURCE, *PRODUCT*
 - soft_same_interval_var, 818
 - soft_same_modulo_var, 820
 - soft_same_partition_var, 822
 - soft_same_var, 824
 - soft_used_by_interval_var, 828
 - soft_used_by_modulo_var, 832
 - soft_used_by_partition_var, 836
 - soft_used_by_var, 838
- NSINK, CC(NSINK, NSOURCE), *PRODUCT*
 - used_by, 930
 - used_by_interval, 934
 - used_by_modulo, 936
 - used_by_partition, 938
- NSINK, NSOURCE, CC(NSINK, NSOURCE), *PRODUCT*
 - same, 754
 - same_interval, 766
 - same_modulo, 768
 - same_partition, 770
- NSINK, NSOURCE, CC(NSINK, NSOURCE), *PRODUCT*; NARC, *PATH*
 - sort, 842
- NSINK, NSOURCE, CC(NSINK, NSOURCE), *PRODUCT*; NVERTEX, *SELF*, \forall

- same_and_global_cardinality, 760
- NSINK, NSOURCE, *PRODUCT*
 - common, 332
 - common_interval, 336
 - common_modulo, 338
 - common_partition, 340
 - in_same_partition, 542
- NSOURCE, *PRODUCT*
 - elements_sparse, 486
- NSOURCE, SUM, *PRODUCT*
 - sum_of_weights_of_distinct_values, 876
- NTREE, *CLIQUE*
 - derangement, 418
- NTREE, SUM_WEIGHT_ARC, *CLIQUE*
 - minimum_weight_alldifferent, 666
- NVERTEX, *PRODUCT*
 - elements_alldifferent, 482
- NVERTEX, *SELF*, \forall
 - global_cardinality, 496
 - global_cardinality_low_up, 500
- NVERTEX, *SELF*, \forall ; SUM_WEIGHT_ARC, *PRODUCT*
 - global_cardinality_with_costs, 502
- ORDER, *CLIQUE*
 - max_index, 624
 - max_n, 626
 - maximum, 634
 - maximum_modulo, 638
 - min_index, 640
 - min_n, 644
 - minimum, 652
 - minimum_except_0, 656
 - minimum_modulo, 664
- PATH_FROM_TO, *CLIQUE*
 - path_from_to, 726
- PATH_FROM_TO, *PRODUCT(PATH, VOID)*
 - lex_greater, 598
 - lex_greatereq, 602

- lex_less, 606
- lex_lesseq, 610
- PATH, LOOP, CC*
- sliding_card_skip0, 786
- PREDEFINED**
- allperm, 204
- colored_matrix, 322
- eq_set, 490
- in_set, 546
- lex2, 580
- pattern, 730
- period, 736
- period_except_0, 738
- set_value_precede, 776
- strict_lex2, 862
- PRODUCT, \forall , SUCC*
- indexed_sum, 552
- PRODUCT*, *SELF*
- product_ctr, 748
- PRODUCT, SUCC*
- assign_and_counts, 234
- assign_and_nvalues, 238
- bin_packing, 264
- interval_and_count, 560
- interval_and_sum, 564
- RANGE_NSCC*, *CLIQUE*
- balance, 250
- balance_interval, 252
- balance_modulo, 256
- balance_partition, 260
- RANGE*, *SELF*
- range_ctr, 750
- SUM_WEIGHT_ARC*, *PRODUCT*
- sliding_time_window_from_start, 798
- SUM*, *PRODUCT*
- sum, 870
- SUM*, *SELF*
- sum_ctr, 874
- sum_set, 880

similarity, **284**

Simonis H., 342, 374

size_maximal_sequence_alldiff, **782**

size_maximal_sequence_alldifferent, 29, 44, 75, 87, 100, 103, 112, 178, **782**, 784

size_maximal_sequence_alldistinct, **782**

size_maximal_starting_sequence_alldiff, **784**

size_maximal_starting_sequence_alldifferent, 28, 75, 87, 100, 103, 112, 178, 782, **784**

size_maximal_starting_sequence_alldistinct, **784**

sliding cyclic(1) constraint network(1), **101**, 411, 549, 684, 686, 688, 864, 866

sliding cyclic(1) constraint network(2), **101**, 285, 290, 404, 407, 416, 534, 555, 734, 808, 942

sliding cyclic(1) constraint network(3), **102**, 285, 290, 620

sliding cyclic(2) constraint network(2), **102**, 300, 449

sliding sequence constraint, **103**, 223, 232, 392, 731, 753, 782, 784, 789, 791, 793, 796, 799, 804, 856, 860

sliding_card_skip0, 63, 66, 68, 100, 103, 106, 215, **786**

sliding_distribution, 80, 87, 100, 103, 500, 501, 730, **790**, 856, 860

sliding_sum, 28, 80, 87, 100, 103, 105, 753, **792**, 875

sliding_time_window, 103, 106, 780, **794**, 798, 799, 802, 803

sliding_time_window_from_start, 81, 103, 106, 796, **798**

sliding_time_window_sum, 103, 105–107, 796, **802**, 875

Smith B., 708

Smith B.M., 508

Smith L.M., 172, 422

smooth, 66, 68, 94, 101, 106, 285, **806**

Soffa M.L., 382

soft constraint, **103**, 753, 811, 815, 819, 821, 823, 825, 829, 833, 837, 839, 948

soft_alldiff_ctr, **810**

soft_alldiff_var, **814**

soft_alldifferent_ctr, 62, 80–82, 84, 99, 103, 108, 178, 179, **810**, 815

soft_alldifferent_var, 62, 81, 82, 84, 99, 103, 104, 108, 109, 178, 179, 811, **814**, 948

`soft_alldistinct_ctr`, **810**
`soft_alldistinct_var`, **814**
`soft_gcc`, 58
`soft_regular`, 58
`soft_same`, **824**
`soft_same_interval`, **818**
`soft_same_interval_var`, 76, 88, 99, 103, 109, **818**
`soft_same_modulo`, **820**
`soft_same_modulo_var`, 76, 92, 99, 103, 109, **820**
`soft_same_partition`, **822**
`soft_same_partition_var`, 76, 96, 99, 103, 109, 543, **822**
`soft_same_var`, 37, 76, 99, 103, 109, 819, 821, 823, **824**
`soft_used_by`, **838**
`soft_used_by_interval`, **828**
`soft_used_by_interval_var`, 76, 88, 99, 103, 109, **828**
`soft_used_by_modulo`, **832**
`soft_used_by_modulo_var`, 76, 92, 99, 103, 109, **832**
`soft_used_by_partition`, **836**
`soft_used_by_partition_var`, 76, 96, 99, 103, 109, 543, **836**
`soft_used_by_var`, 76, 99, 103, 109, **838**
Soriano P., 882
`sort`, **103**, 844, 848
`sort`, 42, 76, 97, 104, 112, 756, **842**, **846**, 848
`sort_permutation`, 13, 76, 81, 97, 104, 178, 346, 348, 844, **846**
sparse functional dependency, **104**, 477, 487
sparse table, **104**, 477, 487
sport timetabling, **104**, 883
squared squares, **104**, 365, 428
Sriskandarajah C., 310

Stadler P.F., 772

stage_element, 66, 68, 70, 73, 79, 85, 106, 459, 462, **850**

stage_elt, **850**

Stergiou K., 508

Stille W., 264

stretch, 113, **854, 858**

stretch_circuit, 79, 103, 106, **854, 860**

stretch_path, 103, 106, 290, 523, 525, 730, 856, **858**

strict_lex2, 89, 91, 95, 98, 105, 580, **862**

strictly_decreasing, 66, 68, 80, 95, 101, 411, 549, **864, 866**

strictly_increasing, 66, 68, 80, 95, 101, 411, 549, 864, **866**

strongly connected component, **104**, 343, 387, 396, 400, 530, 671, 674, 682, 694, 696, 700, 704, 706, 746, 815, 868

strongly_connected, 77, 86, 89, 104, 546, 615, **868**

SUCC, **48**, 234, 238, 264, 324, 329, 362, 366, 371, 375, 379, 552, 560, 564, 660, 676, 680, 794, 802, 900

SUM, **41**, 870, 874, 876, 880, 946

sum, **105**, 793, 804, 871, 875, 880

sum, 78, 79, 89, 105, 546, **870, 875, 880**

sum_ctr, 24, 42, 49, 50, 64, 105, 264, 553, 566, 748, 750, 753, 792, 871, **874, 880**

sum_of_weights_of_distinct_values, 65, 78, 82, 84, 99, 110, **876, 948**

sum_set, 64, 70, 77, 105, 546, 871, 875, **880**

SUM_WEIGHT_ARC, **42**, 503, 666, 798

swc, **760**

swdv, **876**

sweep, **105**, 428

Swinkels G.M., 754, 842

symm_alldiff, **882**

symm_alldifferent, **882**

symm_alldistinct, **882**

- symmetric, **105**, 344
- symmetric_alldiff, **882**
- symmetric_alldifferent, 62, 74, 79, 81, 86, 91, 97, 104, 106, 161, 179, 387, **882**
- symmetric_alldistinct, **882**
- symmetric_cardinality, 65, 77, 80, 84, 99, 106, 498, 546, **886**, 891
- symmetric_gcc, 24, 65, 77, 80, 84, 99, 106, 113, 498, 546, 615, 888, **890**
- SYMMETRIC_PRODUCT*, **29**
- SYMMETRIC_PRODUCT(=)*, 924
- SYMMETRIC_PRODUCT(Comparison)*, **29**
- symmetry, **105**, 204, 556, 558, 580, 586, 589, 593, 599, 603, 607, 611, 776, 862
- Szymanek R., 426

- table, **106**, 459, 462, 465, 469, 477, 480, 484, 487, 577, 677, 681, 851
- Tallys H. Yunes, 870
- Tamassia R., 914

- temporal constraint, **106**, 326, 330, 365, 368, 377, 381, 442, 562, 566, 780, 796, 799, 804, 901
- temporal_path, 75, 86, 96, 727, **892**

- ternary constraint, **106**, 475

- Thiel A.J., 172, 422
- Thiel S., 176, 426, 436, 496, 670, 674, 682, 698, 754, 760, 842, 876, 882, 930, 946
- Thorsteinsson E., 464, 468

- time window, **107**, 804

- timetabling constraint, **106**, 285, 290, 300, 304, 316, 323, 404, 407, 523, 530, 562, 566, 620, 731, 736, 738, 780, 789, 808, 856, 860, 883, 888, 891, 901

- Tollis I.G., 914

- Toth P., 264

- touch, **107**, 724, 920

- tour, 35, 36, 77, 86, 87, 89, 107, 307, 546, 615, **896**

- track, 81, 99, 106, 700, **900**

- tree, **107**, 270, 711, 903, 907, 911

- tree, 35, 37, 75, 86, 107, 268, 270, 387, 514, 623, 711, **902**, 906, 907, 910, 911

- `tree_range`, 38, 69, 75, 86, 107, 250, **906**
- `tree_resource`, 75, 81, 86, 99, 107, 113, 903, **910**
- Trick M.A., 882
- TRUE, **24**
- tuple, **107**, 539, 945
- `two_layer_edge_crossing`, 81, 85, 89, 360, 514, **914**
- `two_orth_are_in_contact`, 66, 68, 69, 77, 85, 94, 96, 107, 717, 724, **918**
- `two_orth_column`, 85, 86, 96, 98, 430, 717, **922**
- `two_orth_do_not_overlap`, 26, 29, 66, 68, 69, 85, 94, 96, 428, 717, **924**
- `two_orth_include`, 85, 96, 98, 432, 717, **928**

- unary constraint, **107**, 537, 691
- undirected graph, **107**, 897
- `used_by`, 66, 67, 72, 76, 84, 88, 756, 838, 839, **930**, 934–939
- `used_by_interval`, 76, 88, 828, 829, **934**
- `used_by_modulo`, 76, 88, 92, 832, 833, **936**
- `used_by_partition`, 76, 88, 96, 543, 836, 837, **938**

- valley, 66, 68, 100, 101, 414, 416, 555, 684, 686, 734, **940**
- value constraint, **108**, 174, 179, 184, 188, 192, 195, 198, 207, 210, 213, 215, 220, 225, 230, 244, 248, 250, 254, 258, 261, 274, 278, 281, 352, 356, 424, 435, 438, 494, 498, 501, 537, 543, 546, 615, 630, 632, 648, 650, 688, 691, 696, 762, 811, 815, 945
- value partitioning constraint, **108**, 671, 674, 682, 694, 700, 704, 706
- value precedence, **109**, 556, 558, 776
- van Beek P., 176, 496, 854, 858
- Van Emden M.H., 754, 842
- Van Hentenryck P., 460, 496
- van Hoeve W.-J., 434, 496, 754, 810, 824
- variable indexing, **109**, 459, 462, 465, 469, 477, 553
- variable subscript, **109**, 459, 462, 465, 469, 553
- variable-based violation measure, **109**, 815, 819, 821, 823, 825, 829, 833, 837, 839
- `vec_eq_tuple`, 107, 108, **944**

vector, **109**, 173, 424, 582, 586, 589, 593, 596, 599, 603, 607, 611

Vilím P., 444

VOID, **29**

vpartition, **110**, 523

Walsh T., 508, 580, 598, 606, 610, 698, 754, 862

Wang C.C., 382

Wei W., 708

weighted assignment, **110**, 504, 668, 877, 948

weighted_partial_alldiff, 62, 65, 78, 88, 99, 103, 110, 178, 179, 184, 504, 668, 815, 877, **946**

weighted_partial_alldifferent, **946**

weighted_partial_alldistinct, **946**

Weihe K., 264

workload covering, **110**, 381

wpa, **946**

Yan H., 362

Zhou J., 842, 846

Zhou N.-F., 342