Global Constraint Catalog

Nicolas Beldiceanu, Mats Carlsson, Jean-Xavier Rampon

Global Constraint Catalog

Nicolas Beldiceanu[1]
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

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

Contents

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

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 checker[1]. 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 trend[2] 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.

[1] A *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.

[2] This 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*:

- `list`(T) corresponds to a list of elements of type T, where T is a basic or a compound data type.

- `c : collection`$(A_1, A_2, \ldots, A_n)$ corresponds to a collection `c` of ordered items, where each item consists of $n > 0$ attributes $A_1, A_2, \ldots, A_n$. Each attribute is an expression of the form $\mathtt{a} - T$, 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 implicit[3] 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* $e_1, e_2, \ldots, e_n$ is denoted $[e_1, e_2, \ldots, e_n]$.

- A *finite set of integers* $i_1, i_2, \ldots, i_n$ is denoted $\{i_1, i_2, \ldots, i_n\}$.

- A *multiset of integers* $i_1, i_2, \ldots, i_n$ is denoted $\{\{i_1, i_2, \ldots, i_n\}\}$.

- A *collection of* n *items*, each item having m attributes, is denoted by $\{\mathtt{a}_1 - v_{11} \ldots \mathtt{a}_m - v_{1m},\ \mathtt{a}_1 - v_{21} \ldots \mathtt{a}_m - v_{2m},\ \ldots\ , \mathtt{a}_1 - v_{n1} \ldots \mathtt{a}_m - v_{nm}\}$. Each item is separated from the previous item by a comma.

- The i^{th} *item of a collection* `c` is denoted $\mathtt{c}[i]$.

- The *number of items* of a collection `c` is denoted $|\mathtt{c}|$.

[3] This 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 is 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.

[a] An *orthotope* corresponds to the generalization of a segment, a rectangle and a box to the n-dimensional case.

[b] 1 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 ▷, which satisfies the restriction is given. Similarly, a ground instance, preceded by the symbol ▶, 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, $[=, \neq, <, \geq, >, \leq]$) forces the last argument CTR of the change constraint to take its value in the list of atoms $[=, \neq, <, \geq, >, \leq]$.
$\triangleright$ change$(1, \{\texttt{var}-4, \texttt{var}-4, \texttt{var}-4, \texttt{var}-6\}, \neq)$
$\blacktriangleright$ change$(1, \{\texttt{var}-4, \texttt{var}-4, \texttt{var}-4, \texttt{var}-6\}, \mathbf{3})$

- 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 $[2, 3, 6]$.
$\triangleright$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 $- 0\}$)
$\blacktriangleright$one_tree($\{$ id $- a$ index $- 1$ type $- \mathbf{9}$ 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).

$\triangleright$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}, $\leq$)

$\blacktriangleright$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}, $\leq$)

- 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.

$\triangleright$cycle(2, {index $-$ 1 succ $-$ 2, index $-$ 2 succ $-$ 1, index $-$ 3 succ $-$ 3})

$\blacktriangleright$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 $\langle v_{i,1}, v_{i,2}, \ldots, v_{i,m} \rangle$ 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 $t_1, t_2, \ldots, t_n$.

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}}

[a] ORTHOTOPES corresponds to the third item of the example presented at page 5.

- $\texttt{Term}_1$ Comparison $\texttt{Term}_2$:
  - $\texttt{Term}_1$ 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 $\leq$, $\geq$, $<$, $>$, $=$, $\neq$.
  - $\texttt{Term}_2$ is a *term*.

Let $v_{1,1}, v_{1,2}, \ldots, v_{1,n_1}$ and $v_{2,1}, v_{2,2}, \ldots, v_{2,n_2}$ be the values respectively associated with $\texttt{Term}_1$ and with $\texttt{Term}_2$. The restriction $\texttt{Term}_1$ `Comparison` $\texttt{Term}_2$ 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 $\texttt{atleast}(\texttt{N}, \texttt{VARIABLES}, \texttt{VALUE})$ constraint: $\texttt{N} \leq |\texttt{VARIABLES}|$ restricts `N` to be less than or equal to the number of items of the `VARIABLES` collection.
$\triangleright \texttt{atleast}(2, \{\texttt{var} - 5, \texttt{var} - 8, \texttt{var} - 5\}, 5)$
$\blacktriangleright \texttt{atleast}(\mathbf{4}, \{\texttt{var} - 5, \texttt{var} - 8, \texttt{var} - 5\}, 5)$

- $\texttt{min\_size}(c, a)$, where c is an argument of type `collection` and a an attribute of c of type `collection`. The value of $\texttt{min\_size}(c, a)$ 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 $\texttt{in\_relation}(\texttt{VARIABLES}, \texttt{TUPLES\_OF\_VALS})$ constraint: $\texttt{min\_size}(\texttt{TUPLES\_OF\_VALS}, \texttt{tuple}) = |\texttt{VARIABLES}|$ forces the smallest number of items associated with the `tuple` attribute to equal the number of items of the `VARIABLES` collection.
$\triangleright \texttt{in\_relation}(\{\{\texttt{var} - 5,\ \texttt{var} - 3,\ \texttt{var} - 3\},$
$\{\texttt{tuple} - \{\texttt{val} - 5,\ \texttt{val} - 2,\ \texttt{val} - 3\},$
$\texttt{tuple} - \{\texttt{val} - 5,\ \texttt{val} - 2,\ \texttt{val} - 6\},$
$\texttt{tuple} - \{\texttt{val} - 5,\ \texttt{val} - 3,\ \texttt{val} - 3\}\})$
$\blacktriangleright \texttt{in\_relation}(\{\{\texttt{var} - 5,\ \texttt{var} - 3,\ \texttt{var} - 3\},$
$\{\mathbf{tuple} - \{\texttt{val} - 5,\ \texttt{val} - 2\},$
$\texttt{tuple} - \{\texttt{val} - 5,\ \texttt{val} - 2,\ \texttt{val} - 6\},$
$\texttt{tuple} - \{\texttt{val} - 5,\ \texttt{val} - 3,\ \texttt{val} - 3\}\})$

- $\texttt{max\_size}(c, a)$, where c is an argument of type `collection` and a an attribute of c of type `collection`. The value of $\texttt{max\_size}(c, a)$ 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 $\texttt{in\_relation}(\texttt{VARIABLES}, \texttt{TUPLES\_OF\_VALS})$ constraint: $\texttt{max\_size}(\texttt{TUPLES\_OF\_VALS}, \texttt{tuple}) = |\texttt{VARIABLES}|$ forces the largest number of items associated with the `tuple` attribute to equal the number of items of the `VARIABLES` collection.
$\triangleright \texttt{in\_relation}(\{\{\texttt{var} - 5,\ \texttt{var} - 3,\ \texttt{var} - 3\},$
$\{\texttt{tuple} - \{\texttt{val} - 5,\ \texttt{val} - 2,\ \texttt{val} - 3\},$
$\texttt{tuple} - \{\texttt{val} - 5,\ \texttt{val} - 2,\ \texttt{val} - 6\},$
$\texttt{tuple} - \{\texttt{val} - 5,\ \texttt{val} - 3,\ \texttt{val} - 3\}\})$
$\blacktriangleright \texttt{in\_relation}(\{\{\texttt{var} - 5,\ \texttt{var} - 3,\ \texttt{var} - 3\},$
$\{\mathbf{tuple} - \{\texttt{val} - 5,\ \texttt{val} - 2,\ \texttt{val} - 8,\ \texttt{val} - 2\},$
$\texttt{tuple} - \{\texttt{val} - 5,\ \texttt{val} - 2,\ \texttt{val} - 6\},$
$\texttt{tuple} - \{\texttt{val} - 5,\ \texttt{val} - 3,\ \texttt{val} - 3\}\})$

– 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 $\texttt{atleast}(\texttt{N}, \texttt{VARIABLES}, \texttt{VALUE})$ constraint: $\texttt{N} \geq 0$ forces the first argument of the `atleast` constraint to be greater than or equal to 0.
$\triangleright \texttt{atleast}(2, \{\texttt{var} - 5, \texttt{var} - 8, \texttt{var} - 5\}, 5)$
$\blacktriangleright \texttt{atleast}(-\mathbf{1}, \{\texttt{var} - 5, \texttt{var} - 8, \texttt{var} - 5\}, 5)$

– 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 $\texttt{among}(\texttt{NVAR}, \texttt{VARIABLES}, \texttt{VALUES})$ constraint: $\texttt{NVAR} \geq 0$ forces the first argument of the `among` constraint to be greater than or equal to 0.
$\triangleright \texttt{among}(2, \{\texttt{var} - 5, \texttt{var} - 8, \texttt{var} - 5\}, \{\texttt{val} - 1, \texttt{val} - 5\})$
$\blacktriangleright \texttt{among}(-\mathbf{9}, \{\texttt{var} - 5, \texttt{var} - 8, \texttt{var} - 5\}, \{\texttt{val} - 1, \texttt{val} - 5\})$

– $c.a$, 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 $\texttt{cumulative}(\texttt{TASKS}, \texttt{LIMIT})$ constraint: $\texttt{TASKS.duration} \geq 0$ enforces for all items of the `TASKS` collection that the `duration` attribute be greater than or equal to 0.

$$\triangleright \texttt{cumulative}(\{\ \texttt{origin} - 2\ \texttt{duration} - 2\ \texttt{end} - 4\ \texttt{height} - 2,\ \texttt{origin} - 2\ \texttt{duration} - 2\ \texttt{end} - 4\ \texttt{height} - 2,\ \texttt{origin} - 1\ \texttt{duration} - 4\ \texttt{end} - 5\ \texttt{height} - 5,\ \texttt{origin} - 4\ \texttt{duration} - 2\ \texttt{end} - 6\ \texttt{height} - 1\},\ 12)$$

$$\blacktriangleright \texttt{cumulative}(\{\ \texttt{origin} - 2\ \texttt{duration} - -\mathbf{2}\ \texttt{end} - 4\ \texttt{height} - 2,\ \texttt{origin} - 2\ \texttt{duration} - 2\ \texttt{end} - 4\ \texttt{height} - 2,\ \texttt{origin} - 1\ \texttt{duration} - 4\ \texttt{end} - 5\ \texttt{height} - 5,\ \texttt{origin} - 4\ \texttt{duration} - 2\ \texttt{end} - 6\ \texttt{height} - 1\},\ 12)$$

– $c.a$, 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.

[4] This 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 ≥ 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 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 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: ATMOST $\leq$ |VARIABLES| − 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 $\texttt{ctr}(A_1, A_2, \ldots, A_n)$, where ctr corresponds to the *name* of the global constraint and $A_1, A_2, \ldots, A_n$ 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* $A_1{:}T_1, A_2{:}T_2, \ldots, A_n{:}T_n$ giving for each argument $A_1, A_2, \ldots, A_n$ 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	all_differ_from_at_least_k_pos(K, VECTORS)
Type(s)	VECTOR − collection(var − dvar)
Argument(s)	K − int
	VECTORS − collection(vec − VECTOR)
Restriction(s)	required(VECTOR, var)
	$\texttt{K} \geq 0$
	required(VECTORS, vec)
	same_size(VECTORS, vec)

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.

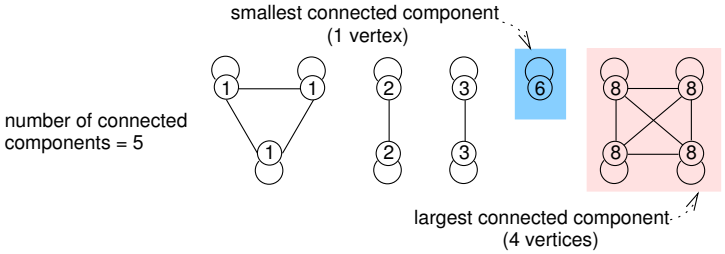


Figure 1.1: Illustration of the link between graph-properties and global constraints

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 been 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 $\texttt{nvalue}(\texttt{NVAL}, \texttt{VARIABLES})$ where NVAL and VARIABLES respectively correspond to a domain variable and to a collection of domain variables $\{\texttt{var} - V_1, \texttt{var} - V_2, \ldots, \texttt{var} - V_m\}$[6]. This constraint holds if NVAL is equal to the number of distinct values assigned to the variables $V_1, V_2, \ldots, V_m$. 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 $(\texttt{NVAL}, \{\texttt{var} - V_1, \texttt{var} - V_2, \texttt{var} - V_3, \texttt{var} - V_4\})$, 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 $\texttt{nvalue}(3, \{\texttt{var} - 5, \texttt{var} - 5, \texttt{var} - 1, \texttt{var} - 8\})$. 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 $\texttt{nvalue}(3, \{\texttt{var}-5, \texttt{var}-5, \texttt{var}-1, \texttt{var}-8\})$ 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.

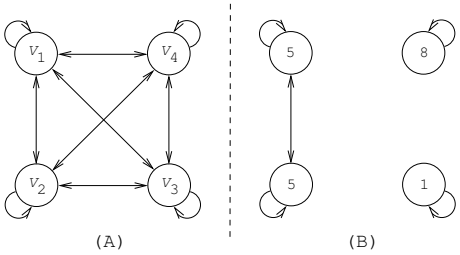


Figure 1.2: Initial and final graph associated with `nvalue`

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 $\texttt{nvalue}(\texttt{NVAL}, \texttt{VARIABLES})$ 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.

- *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 $\texttt{ITEM} - \texttt{collection}(\texttt{index} - \texttt{dvar}, \texttt{value} - \texttt{dvar})$ 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 $o - \texttt{item}(\texttt{a}_1 - v_1, \texttt{a}_2 - v_2, \ldots, \texttt{a}_n - v_n)$ or $\texttt{item}(\texttt{a}_1 - v_1, \texttt{a}_2 - v_2, \ldots, \texttt{a}_n - v_n)$ specifies for each attribute $\texttt{a}_i (1 \leq i \leq n)$ of the new collection how to fill it[7]. This is done by providing for each attribute $\texttt{a}_i$ one of the following element v_i:

[7] o is one of the comparison operators $=, \neq, <, \geq, >, \leq$. 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 $o - \texttt{item}(\texttt{a}_1 - v_1, \texttt{a}_2 - v_2, \ldots, \texttt{a}_n - v_n)$ does not contain any reference to an attribute of a collection then we generate one single item for such pattern[8]. In this context the value v_i of the attribute $\texttt{a}_i$ $(1 \leq i \leq n)$ corresponds to a constant, to an argument of the global constraint or to a new derived collection.

- If the pattern $o - \texttt{item}(\texttt{a}_1 - v_1, \texttt{a}_2 - v_2, \ldots, \texttt{a}_n - v_n)$, where o is one of the comparison operators $=, \neq, <, \geq, >, \leq$, contains one or several references to an attribute of a collection[9] we denote by:

  - $k_1, k_2, \ldots, k_m$ the indices of the positions corresponding to the attribute of a collection within $\texttt{item}(\texttt{a}_1 - v_1, \texttt{a}_2 - v_2, \ldots, \texttt{a}_n - v_n)$,
  - $\texttt{c}_{\alpha_1}, \texttt{c}_{\alpha_2}, \ldots, \texttt{c}_{\alpha_m}$ the corresponding collections,
  - $\texttt{a}_{\alpha_1}, \texttt{a}_{\alpha_2}, \ldots, \texttt{a}_{\alpha_m}$ the corresponding attributes.

  For each combination of items $\texttt{c}_{\alpha_1}[i_1], \texttt{c}_{\alpha_2}[i_2], \ldots, \texttt{c}_{\alpha_m}[i_m]$ such that:

  $$i_1 \in [1, |\texttt{c}_{\alpha_1}|], i_2 \in [1, |\texttt{c}_{\alpha_2}|], \ldots, i_m \in [1, |\texttt{c}_{\alpha_m}|] \text{ and } i_1 \; o \; i_2 \; o \; \ldots \; o \; i_n$$

  we generate an item of the new derived collection $(\texttt{a}_1 - w_1 \; \texttt{a}_2 - w_2 \; \ldots \; \texttt{a}_n - w_n)$ defined by:

  $$w_j (1 \leq j \leq n) = \begin{cases} \texttt{c}_{\alpha_p}[i_p].\texttt{a}_{\alpha_p} & \text{if} j \in \{k_1, k_2, \ldots, k_m\}, j = k_p \\ v_j & \text{if} j \notin \{k_1, k_2, \ldots, k_m\} \end{cases} .$$

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

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

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.key[a], 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. $\forall i_1 \in [1, |\texttt{VECTOR1}|], \forall i_2 \in [1, |\texttt{VECTOR2}|]$ such that $i_1 = i_2$[b] we generate an item index − v_1 x − v_2 y − v_3 where:
$v_1 = i_1$, $v_2 = \texttt{VECTOR1}[i_1].\texttt{var}$, $v_3 = \texttt{VECTOR2}[i_1].\texttt{var}$.
This leads to the four items listed in the GENERATED ITEM(S) field.

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

[b] We 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 − item(...).

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 $- u_1$ duration $- u_2$ point $- u_3$, idm $- v_1$ duration $- v_2$ point $- v_3$ where:

$u_1 =$ TASKS$[i]$.machine, $u_2 =$ TASKS$[i]$.duration, $u_3 =$ TASKS$[i]$.origin,
$v_1 =$ TASKS$[i]$.machine, $v_2 =$ TASKS$[i]$.duration, $v_3 =$ TASKS$[i]$.end.

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. $\forall i_1 \in [1, |\texttt{VARIABLES}|]$, $\forall i_2 \in [1, |\texttt{VARIABLES}|]$ such that $i_1 > i_2$[a] we generate the item $\texttt{x} - u_1\ \texttt{y} - u_2$ where:
$u_1 = \texttt{VARIABLES}[i_1].\texttt{var}$, $u_2 = \texttt{VARIABLES}[i_2].\texttt{var}$.
This leads to the six items listed in the GENERATED ITEM(S) field.

[a] We use the comparison operator $>$ since we have a pattern of the form $> -\texttt{item}(\ldots)$.

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 generator[10] 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.

[10] Arc generators are described in Section 1.2.2 (page 26).
[11] The 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 $SELF \mapsto$ collection(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 Expr^{th} 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].

[a] This 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.

[b] This 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*.
- Exp_1 Op Exp_2:
  - Exp_1 is an *arithmetic expression*,
  - Op is one of the following symbols $+$, $-$, $*$, / [12],
  - Exp_2 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.

- |Exp|:
  - Exp is an *arithmetic expression*, and |Exp| denotes the absolute value of this expression.
- sign(Exp):
  - Exp is an *arithmetic expression*, and sign(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:
$\texttt{sign}((\texttt{s2.ox} - \texttt{s1.ex}) * (\texttt{s1.ey} - \texttt{s1.oy}) - (\texttt{s1.ex} - \texttt{s1.ox}) * (\texttt{s2.oy} - \texttt{s1.ey})) \neq \texttt{sign}((\texttt{s2.ex} - \texttt{s1.ex}) * (\texttt{s2.oy} - \texttt{s1.oy}) - (\texttt{s2.ox} - \texttt{s1.ox}) * (\texttt{s2.ey} - \texttt{s1.ey}))$

- card_set(Set):
  - Set is a reference to a set of integers or to a set variable. card_set(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: $\texttt{vars.nocc} = \texttt{card\_set}(\texttt{vars.var})$.

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

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

- $\mathtt{Exp}_1$ SimpleCtr $\mathtt{Exp}_2$:
  - $\mathtt{Exp}_1$ is an *arithmetic expression*,
  - SimpleCtr is an argument of type atom that can only take one of the values $\leq, \geq, <, >, =, \neq$,
  - $\mathtt{Exp}_2$ 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.

- $\mathtt{Exp}_1$ $\neg$SimpleCtr $\mathtt{Exp}_2$:
  - $\mathtt{Exp}_1$ is an *arithmetic expression*,
  - SimpleCtr is an argument of type atom that can only take one of the values $\leq, \geq, <, >, =, \neq$,
  - $\mathtt{Exp}_2$ 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 $\neg$CTR variables2.var. Within this expression, variables1 and variables2 correspond to consecutive items of the VARIABLES collection.

- Ctr($\mathtt{Exp}_1, \ldots, \mathtt{Exp}_n$):
  - Ctr is a global constraint defined in the catalog for which there exists a graph-based and/or an automaton-based representation,
  - $\mathtt{Exp}_1, \ldots, \mathtt{Exp}_n$ 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(ORTHOTOPES) uses the two_orth_do_not_overlap(ORTHOTOPE1, ORTHOTOPE2) global constraint for defining its arc constraint. Since ORTHOTOPES is a collection of type collection(ori − dvar, siz − dvar, end − dvar) and since both ORTHOTOPE1 and ORTHOTOPE2 correspond to items of ORTHOTOPES there is no type compatibility problem between the call to two_orth_do_not_overlap and its definition.

- $\texttt{ArcCtr}_1$ LogicalConnector $\texttt{ArcCtr}_2$:
  - $\texttt{ArcCtr}_1$ is an *arc constraint*,
  - LogicalConnector is one of the logical connectors $\vee$, $\wedge$, $\Rightarrow$, $\Leftrightarrow$,
  - $\texttt{ArcCtr}_2$ 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 order[13]. 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

[13] Usually 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.

takes n collections of items, denoted $\mathtt{c}_i (1 \leq i \leq n)$, as input and returns the corresponding hypergraph where the vertices are the items of the input collections $\mathtt{c}_i (1 \leq i \leq n)$ 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 $i_1\ i_2\ \ldots\ i_a$ and is denoted by $(i_1, i_2, \ldots, i_a)$. 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 $\mathtt{c}$ and generates the following arcs[14]:

  – $\forall i \in [1, |\mathtt{c}| - 1]$: $(\mathtt{c}[i], \mathtt{c}[i+1])$, – $\forall i \in [1, |\mathtt{c}| - 1]$: $(\mathtt{c}[i+1], \mathtt{c}[i])$.

  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 $\mathtt{c}$ and generates the following arcs:

  – $\forall i \in [1, |\mathtt{c}| - 1]$: $(\mathtt{c}[i], \mathtt{c}[i+1])$, – $(\mathtt{c}[|\mathtt{c}|], \mathtt{c}[1])$.

  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 $\mathtt{c}$ and generates the arcs: $\forall i_1 \in [1, |\mathtt{c}|], \forall i_2 \in [1, |\mathtt{c}|], \ldots, \forall i_a \in [1, |\mathtt{c}|] : (\mathtt{c}[i_1], \mathtt{c}[i_2], \ldots, \mathtt{c}[i_a])$.

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

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

- $CLIQUE(\texttt{Comparison})$, 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:

  $\forall i_1 \in [1, |\texttt{c}|]$,

  $\forall i_2 \in [1, |\texttt{c}|]$ such that i_1 Comparison i_2,

  .....,

  $\forall i_a \in [1, |\texttt{c}|]$ such that i_{a-1} Comparison $i_a : (\texttt{c}[i_1], \texttt{c}[i_2], \ldots, \texttt{c}[i_a])$.

  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.

- $GRID([d_1, d_2, \ldots, d_n])$ takes a collection c consisting of $d_1 \cdot d_2 \cdot \cdots \cdot d_n$ items and generates the arcs $(\texttt{c}[i], \texttt{c}[j])$ 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:

  (1) $|i - j| = \prod_{1 \leq k \leq \alpha} d_k$ (when $\alpha = 0$ we have $\prod_{1 \leq k \leq \alpha} = 1$),

  (2) $\lfloor \frac{i}{\prod_{1 \leq k \leq \alpha+1} d_k} \rfloor = \lfloor \frac{j}{\prod_{1 \leq k \leq \alpha+1} d_k} \rfloor$.

  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: $\forall i \in [1, |\texttt{c}|]$: $(\texttt{c}[i], \texttt{c}[i])$. $LOOP$ 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: $\forall i \in [1, |\texttt{c}| - a + 1] : (\texttt{c}[i], \texttt{c}[i + 1], \ldots, \texttt{c}[i + a - 1])$.

  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: $(\texttt{c}[1])$, $(\texttt{c}[1], \texttt{c}[2])$, ... , $(\texttt{c}[1], \texttt{c}[2], \ldots, \texttt{c}[|\texttt{c}|])$.

  EXAMPLE: $PATH\_1$ is used in the size_maximal_starting_sequence_alldifferent constraint.

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

  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 $\mathtt{c}_1$, $\mathtt{c}_2$ and generates the arcs: $\forall i \in [1, |\mathtt{c}_1|], \forall j \in [1, |\mathtt{c}_2|]$: $(\mathtt{c}_1[i], \mathtt{c}_2[j])$.

  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.

- $PRODUCT(\mathtt{Comparison})$, where `Comparison` is one of the comparison operators $\leq$, $\geq$, $<$, $>$, $=$, $\neq$, has a predefined arity of 2. It takes two collections $\mathtt{c}_1$, $\mathtt{c}_2$ and generates the arcs: $\forall i \in [1, |\mathtt{c}_1|], \forall j \in [1, |\mathtt{c}_2|]$ such that i `Comparison` j : $(\mathtt{c}_1[i], \mathtt{c}_2[j])$.

  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: $\forall i \in [1, |\mathtt{c}|]$: $(\mathtt{c}[i])$.

  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 $\mathtt{c}_1$, $\mathtt{c}_2$ and generates the following arcs: $\forall i \in [1, |\mathtt{c}_1|], \forall j \in [1, |\mathtt{c}_2|]$: $(\mathtt{c}_1[i], \mathtt{c}_2[j])$ and $(\mathtt{c}_2[j], \mathtt{c}_1[i])$. $SYMMETRIC\_PRODUCT$ is currently not used.

- $SYMMETRIC\_PRODUCT(\mathtt{Comparison})$, where `Comparison` is one of the comparison operators $\leq$, $\geq$, $<$, $>$, $=$, $\neq$, has a predefined arity of 2. It takes two collections $\mathtt{c}_1$, $\mathtt{c}_2$ and generates the arcs: $\forall i \in [1, |\mathtt{c}_1|], \forall j \in [1, |\mathtt{c}_2|]$ such that i `Comparison` j : $(\mathtt{c}_1[i], \mathtt{c}_2[j])$ and $(\mathtt{c}_2[j], \mathtt{c}_1[i])$.

  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 $\mathcal{G}enerator = \{CIRCUIT, CHAIN, CLIQUE, LOOP, PATH, VOID\}$. This is achieved by using the construction $PRODUCT(\mathtt{G}_1, \mathtt{G}_2)$ where $\mathtt{G}_1$ and $\mathtt{G}_2$ belong to $\mathcal{G}enerator$. It applies $\mathtt{G}_1$ to the first collection $\mathtt{c}_1$ passed to $PRODUCT$ and $\mathtt{G}_2$ to the second collection $\mathtt{c}_2$ passed to $PRODUCT$. Finally, it applies $PRODUCT$ on $\mathtt{c}_1$ and $\mathtt{c}_2$. In a similar way the $PRODUCT(\mathtt{Comparison})$ arc generator is extended to $PRODUCT$ $(\mathtt{G}_1, \mathtt{G}_2, \mathtt{Comparison})$.

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.

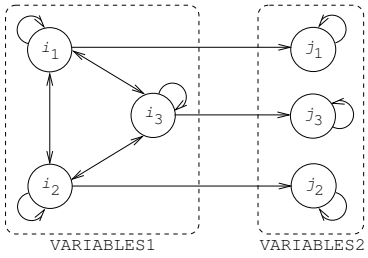


Figure 1.3: Example of initial graph generated by $PRODUCT(CLIQUE, LOOP, =)$

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

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 `Char Comparison Exp` or the form $\texttt{Char} \notin [\texttt{Exp}_1, \texttt{Exp}_2]$, where `Char` is a graph characteristic [17], [31], `Comparison` is one of the comparison operators $=$, $<$, $\geq$, $>$, $\leq$, $\neq$, and `Exp`, $\texttt{Exp}_1$, $\texttt{Exp}_2$ 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 $(e_1, e_2, \ldots, e_k)$ 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.

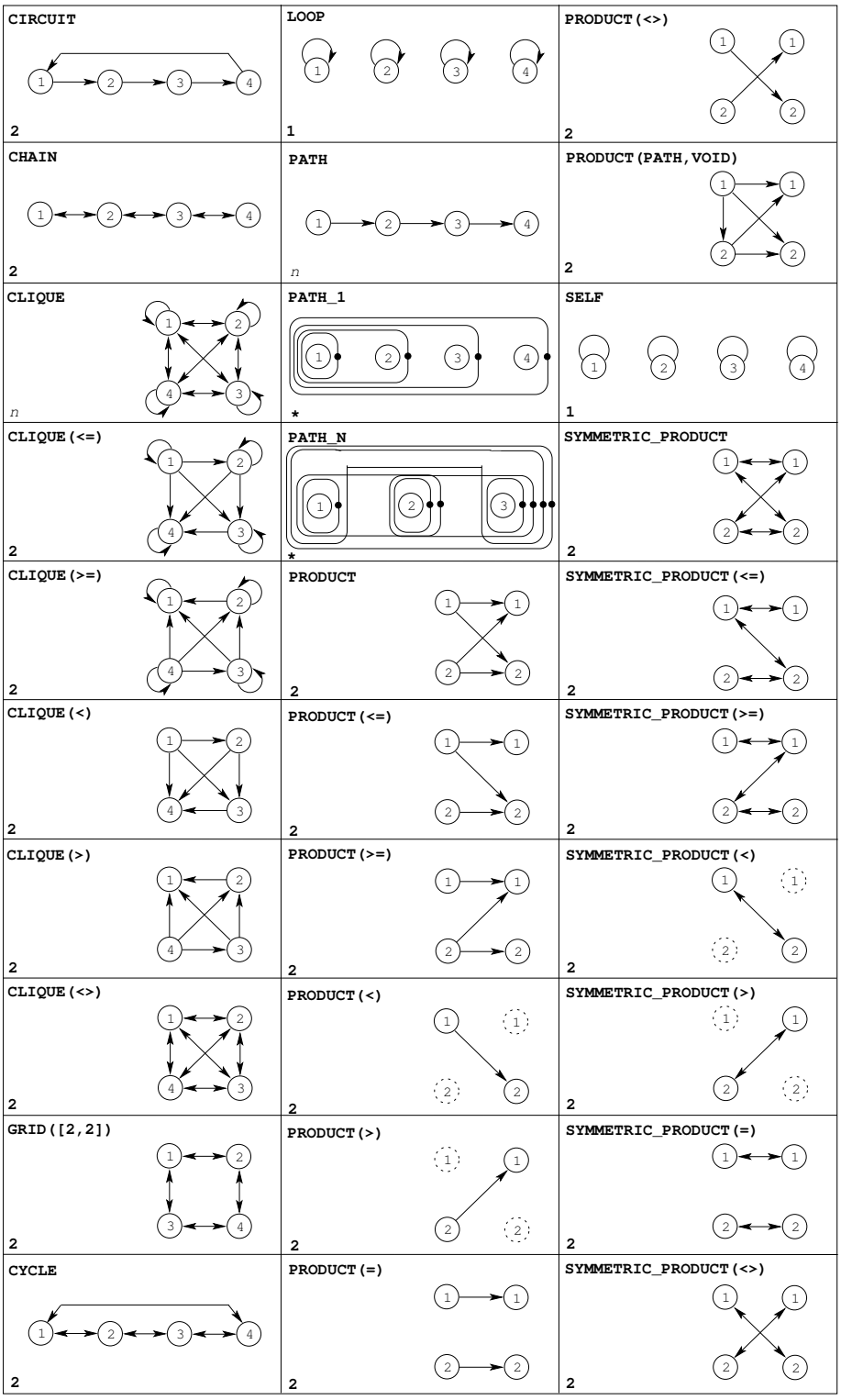


Figure 1.4: Examples of arc generators

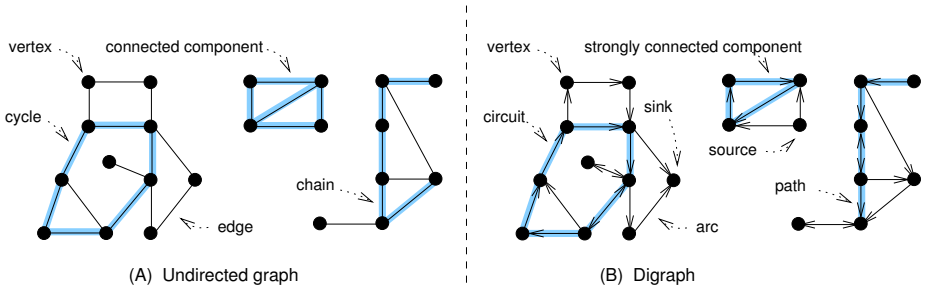


Figure 1.5: Graph terminology for an undirected graph and a digraph

- A sequence $(e_1, e_2, \ldots, e_k)$ 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 $(e_1, e_2, \ldots, e_k)$ 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 $(e_1, e_2, \ldots, e_k)$ 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 $(e_1, e_2, \ldots, e_r)$ 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 $\{1, \cdots, k\}$ for k any positive integer.

- Given a set X, $|X|$ is the number of its elements.

- Given two sets X and Y, $X \biguplus Y$ 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, $\mathbf{NCC}_{\texttt{INITIAL}}$ 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:

- $\mathbf{MAX\_DRG}$: 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 $\mathbf{MAX\_DRG}$ is currently not used.

- $\mathbf{MAX\_ID}$: 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 $\mathbf{MAX\_ID} = 1$ in order to force each vertex of the final graph to have at most one predecessor.

- $\mathbf{MAX\_NCC}$: number of vertices of the largest connected component of G_f.

  EXAMPLE: The `longest_change`(`SIZE`, `VARIABLES`, `CTR`) constraint uses the graph property $\mathbf{MAX\_NCC} =$ `SIZE` in order to catch in `SIZE` the maximum number of consecutive variables of the `VARIABLES` collection for which constraint `CTR` holds.

- $\mathbf{MAX\_NSCC}$: 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 $\mathbf{MAX\_NSCC} \leq 1$ in order to avoid to have any circuit involving more than one vertex.

- $\mathbf{MAX\_OD}$: 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 $\mathbf{MAX\_OD} = 2$ to enforce that each vertex of G_f have at most two[a] successors.

[a]Since the `tour` constraint uses the $CLIQUE(\neq)$ arc generator the vertices of G_f don't have any loop.

- $\mathbf{MIN\_DRG}$: 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 $\mathbf{MIN\_DRG}$ is currently not used by any constraint.

- $\mathbf{MIN\_ID}$: 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 $\mathbf{MIN\_ID} = 2$ to enforce that each vertex of G_f have at most two[a] predecessors.

[a]Since the `tour` constraint uses the $CLIQUE(\neq)$ arc generator the vertices of G_f don't have any loop.

- $\mathbf{MIN\_NCC}$: 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 $\mathbf{MIN\_NCC} =$ `MIN_SIZE` enforces that the smallest sequence of such variables consist of `MIN_SIZE` variables.

- $\mathbf{MIN\_NSCC}$: 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 $\mathbf{MIN\_NSCC} = |\texttt{NODES}|$ enforces that the smallest strongly connected component of G_f contain $|\texttt{NODES}|$ vertices. Since $|\texttt{NODES}|$ 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.

[a] Of course, this is not enough, and the description of the `circuit` constraint asks for some other properties.

- $\mathbf{MIN\_OD}$: 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 $\mathbf{MIN\_OD} = 2$ to enforce that each vertex of G_f have at most two[a] successors.

[a] Since the `tour` constraint uses the $CLIQUE(\neq)$ arc generator the vertices of G_f don't have any loop.

- $\mathbf{NARC}$: cardinality of the set $E(G_f)$.

EXAMPLE: The `disjoint`$(\texttt{VARIABLES1}, \texttt{VARIABLES2})$ 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 $\mathbf{NARC} = 0$ 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`.

- $\mathbf{NARC\_NO\_LOOP}$: 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 $\mathbf{NARC\_NO\_LOOP}$ graph-property.

- $\mathbf{NCC}$: 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 $\mathbf{NCC} = \texttt{NTREES}$ in order to state that G_f is made up from `NTREES` connected components.

- $\mathbf{NSCC}$: number of strongly connected components of G_f.

EXAMPLE: The constraint $\texttt{nvalue}(\texttt{NVAL}, \texttt{VARIABLES})$ 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 $\mathbf{NSCC} = \texttt{NVAL}$. Each strongly connected component of the final graph corresponds to the variables that are assigned to the same value.

- $\mathbf{NSINK}$: number of vertices of G_f that do not have any successor.

EXAMPLE: The $\texttt{same}(\texttt{VARIABLES1}, \texttt{VARIABLES2})$ 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 $\mathbf{NSINK} = |\texttt{VARIABLES2}|$ in order to express the fact that each value assigned to a variable of `VARIABLES2` should also be assigned to a variable of `VARIABLES1`.

- $\mathbf{NSINK\_NSOURCE}$: 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 $\texttt{soft\_same\_var}(\texttt{C}, \texttt{VARIABLES1}, \texttt{VARIABLES2})$ 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 $\mathcal{C}_{val}$ of the final graph G_f corresponds to all variables that are assigned to the same value val: the sources and the sinks of $\mathcal{C}_{val}$ 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 $\mathbf{NSINK\_NSOURCE} = |\texttt{VARIABLES1}| - \texttt{C}$ for encoding the meaning of the `soft_same_var` constraint.

[a] Both collections have the same number of variables.

- $\mathbf{NSOURCE}$: number of vertices of G_f that do not have any predecessor.

EXAMPLE: The $\texttt{same}(\texttt{VARIABLES1}, \texttt{VARIABLES2})$ 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 $\mathbf{NSOURCE} = |\texttt{VARIABLES1}|$ in order to express the fact that each value assigned to a variable of VARIABLES1 should also be assigned to a variable of VARIABLES2.

- $\mathbf{NTREE}$: 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 $\texttt{cycle}(\texttt{NCYCLE}, \texttt{NODES})$ 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 $\mathbf{NTREE} = 0$ enforces that all vertices of the final graph belong to a circuit.

- $\mathbf{NVERTEX}$: cardinality of the set $V(G_f)$.

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

- $\mathbf{RANGE\_DRG}$: 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 $\mathbf{RANGE\_DRG} = \texttt{R}$.

- **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 $\mathtt{var}_1$ to a variable $\mathtt{var}_2$ if and only if $\mathtt{var}_1 < \mathtt{var}_2$. The graph-property **ORDER**$(0, \mathtt{MAXINT}, \mathtt{var}) = \mathtt{MIN}$ expresses the fact that MIN is equal to the value of the source of G_f (since $\mathtt{rank} = 0$).

- **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`). $\mathbf{PATH\_FROM\_TO}(\texttt{attr}, \texttt{from}, \texttt{to})$ 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 $\mathbf{PATH\_FROM\_TO}$ graph-property.

- $\mathbf{PRODUCT}(\texttt{col}, \texttt{attr})$
  - `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, $\mathbf{PRODUCT}(\texttt{col}, \texttt{attr})$ corresponds to the product of the values of attribute `attr` associated with the vertices of $\mathcal{V}$,
  - Otherwise, if $\mathcal{V}$ is empty, $\mathbf{PRODUCT}(\texttt{col}, \texttt{attr})$ 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, $\mathbf{PRODUCT}(\texttt{VARIABLES}, \texttt{var})$ `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 $\mathbf{PRODUCT}(\texttt{VARIABLES}, \texttt{var})$ corresponds to the product of the variables of the `VARIABLES` collection.

- $\mathbf{RANGE}(\texttt{col}, \texttt{attr})$:
  - `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, $\mathbf{RANGE}(\texttt{col}, \texttt{attr})$ 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, $\mathbf{RANGE}(\texttt{col}, \texttt{attr})$ 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, $\mathbf{RANGE}(\texttt{VARIABLES}, \texttt{var})$ 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 $\mathbf{RANGE}(\texttt{VARIABLES}, \texttt{var})$ corresponds to the difference between the maximum value and the minimum value of the variables of the VARIABLES collection.

- $\mathbf{SUM}(\texttt{col}, \texttt{attr})$:

– 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, $\mathbf{SUM}(\texttt{col}, \texttt{attr})$ corresponds to the sum of the values of attribute attr associated with the vertices of $\mathcal{V}$,

– Otherwise, if $\mathcal{V}$ is empty, $\mathbf{SUM}(\texttt{col}, \texttt{attr})$ 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, $\mathbf{SUM}$(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 $\mathbf{SUM}$(VARIABLES, var) corresponds to the sum of the variables of the VARIABLES collection.

- $\mathbf{SUM\_WEIGHT\_ARC}$(Expr) : Expr is an *arithmetic expression*. For each arc a of $E(G_f)$, let $f(a)$ denote the value of Expr. $\mathbf{SUM\_WEIGHT\_ARC}$(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 $((i-1) * |\text{VALUES}| + j)^{th}$ entry of the MATRIX collection.

A last graph characteristic, $\mathbf{DISTANCE}$, 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 restrictions[16]. This section first describes *simple graph constraints* and then *dynamic graph constraints*, which are an extension of *simple graph constraints*.

[15]For 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.

[16]The 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 arc[17] have to be pairwise mutually incompatible[18].

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 $C_1, C_2, \ldots, C_d$ $(d \geq 1)$. To each item of these collections corresponds a vertex of the initial graph.

- An **Arc generator** field, which can be one or several expressions[19] of the following forms:

  - $ARC\_GENERATOR \mapsto \texttt{collection}(\texttt{item}_1, \texttt{item}_2, \ldots, \texttt{item}_a)$, where $ARC\_GENERATOR$ is one of the arc generators with a fixed arity[20] defined in Section 1.2.2 page 26, and $\texttt{item}_i$ $(1 \leq i \leq a)$ denotes the i^{th} item associated with the i^{th} vertex of an arc. These items correspond to formal parameters[21] which can be used within an arc constraint. When the **Arc input(s)** field consists of one single collection $(d = 1)$, $\texttt{item}_i$ $(1 \leq i \leq a)$ represents an item of the collection C_1. Otherwise, when $d > 1$, we must have $a = d$ and, in this context, $\texttt{item}_i$ $(1 \leq i \leq a)$ 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 $CLIQUE \mapsto$ collection (variables1, variables2) (i.e. $a = 2$).

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

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

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

[19] Usually one single expression.

[20] Any arc generator different from $PATH\_1$ and $PATH\_N$.

[21] See 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 $PRODUCT \mapsto$ collection(variables1, 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.

– $ARC\_GENERATOR \mapsto$ 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 $PRODUCT \mapsto$ 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:

$ARC\_GENERATOR_1 \mapsto \texttt{collection}(\texttt{item}_1, \texttt{item}_2, \ldots, \texttt{item}_a)$

$ARC\_GENERATOR_2 \mapsto \texttt{collection}(\texttt{item}_1, \texttt{item}_2, \ldots, \texttt{item}_a)$

.....

$ARC\_GENERATOR_n \mapsto \texttt{collection}(\texttt{item}_1, \texttt{item}_2, \ldots, \texttt{item}_a)$

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.

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

- 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):		$\mathbf{NSCC} =$ 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):		$\mathbf{NCC} \leq 1$

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 itself[a]. 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 $\mathbf{NCC} \leq 1$ enforces to have at most one such group of adjacent variables assigned to 1.

[a] We 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):		$\mathbf{NVERTEX} =$ 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 paths[23] 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.

[23] A 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.

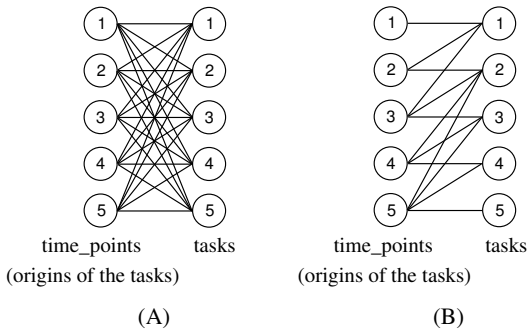


Figure 1.6: Initial and final graph of an instance of the `cumulative` constraint

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

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

We label the vertices of the initial and final graph by giving the $\texttt{key}^a$ 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:

- $\{\texttt{var}-1\}$,
- $\{\texttt{var}-1,\ \texttt{var}-2\}$,
- $\{\texttt{var}-1,\ \texttt{var}-2,\ \texttt{var}-1\}$,
- $\{\texttt{var}-2,\ \texttt{var}-1,\ \texttt{var}-1\}$,
- $\{\texttt{var}-2,\ \texttt{var}-1,\ \texttt{var}-1,\ \texttt{var}-3\}$.

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

- $\texttt{sum\_ctr}(\{\texttt{var}-1\},\ \leq,\ 8)$,
- $\texttt{sum\_ctr}(\{\texttt{var}-1,\ \texttt{var}-2\},\ \leq,\ 8)$,
- $\texttt{sum\_ctr}(\{\texttt{var}-1,\ \texttt{var}-2,\ \texttt{var}-1\},\ \leq,\ 8)$,
- $\texttt{sum\_ctr}(\{\texttt{var}-2,\ \texttt{var}-1,\ \texttt{var}-1\},\ \leq,\ 8)$,
- $\texttt{sum\_ctr}(\{\texttt{var}-2,\ \texttt{var}-1,\ \texttt{var}-1,\ \texttt{var}-3\},\ \leq,\ 8)$.

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

[a] `key` 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],
- `minimum` [33],
- `pattern` [34],
- `global_contiguity` [35],
- `lex_lesseq` [36],
- `among` [37],
- `inflexion` [3],
- `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 $\overrightarrow{x} \leq_{\text{lex}} \overrightarrow{y}$ (see `lex_lesseq`) over two vectors of variables $\overrightarrow{x} = \langle x_0, \ldots, x_{n-1} \rangle$ and $\overrightarrow{y} = \langle y_0, \ldots, y_{n-1} \rangle$ holds iff $n = 0$ or $x_0 < y_0$ or $x_0 = y_0$ and $\langle x_1, \ldots, x_{n-1} \rangle \leq_{\text{lex}} \langle y_1, \ldots, y_{n-1} \rangle$.

- 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.

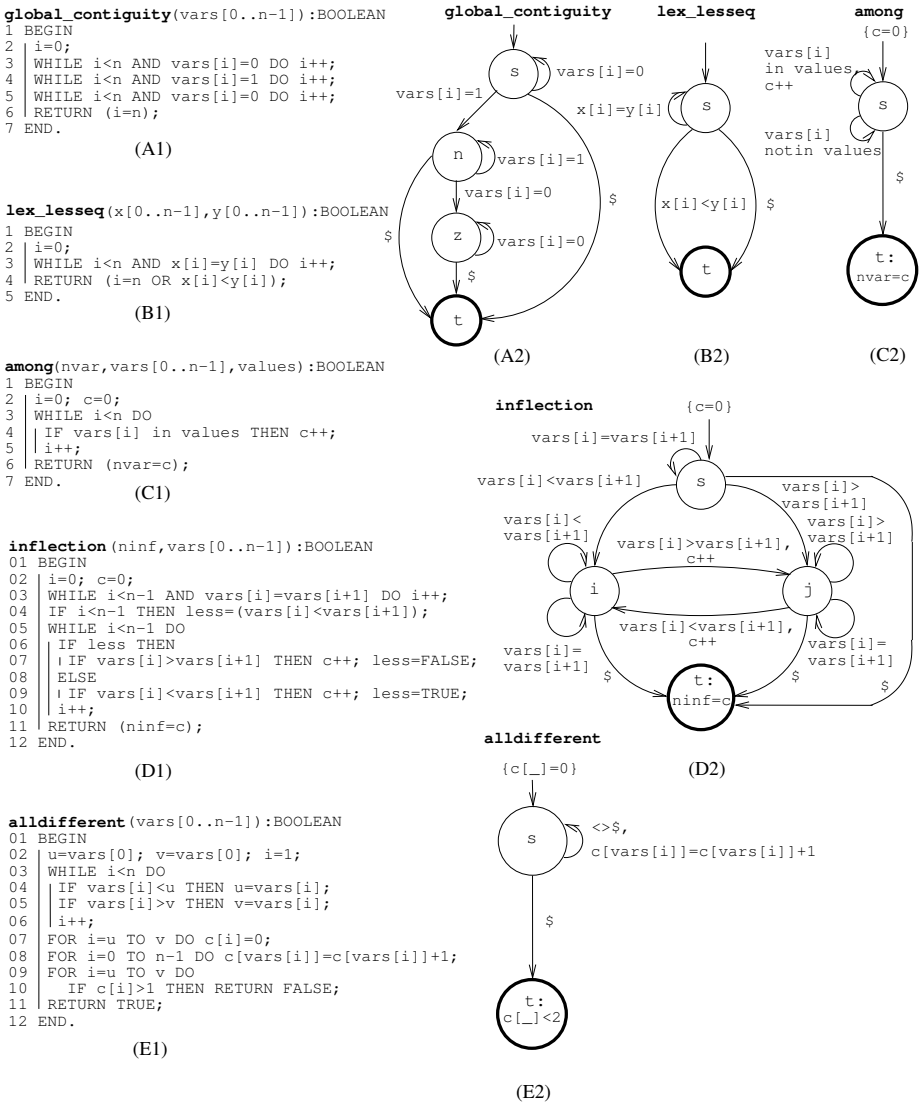


Figure 1.7: Five checkers and their corresponding automata

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 $\texttt{vars}[0], \ldots, \texttt{vars}[n-1]$ 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 $\texttt{vars}[0], \ldots, \texttt{vars}[n-1]$, and triggers successively a transition for each term of this sequence. Transitions labeled by 0, 1 and \$ are respectively associated with the conditions $\texttt{vars}[i] = 0$, $\texttt{vars}[i] = 1$ 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 $\overrightarrow{x}$ and $\overrightarrow{y}$ 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 $\langle x[0], y[0]\rangle, \ldots, \langle x[n-1], y[n-1]\rangle$ 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 $\texttt{vars}[0], \ldots, \texttt{vars}[n-1]$ 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 $\texttt{vars}[0], \ldots, \texttt{vars}[n-1]$. 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 $\texttt{vars}[0]$, $\ldots$, $\texttt{vars}[n-1]$. 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

$\langle \texttt{vars}[0], \texttt{vars}[1] \rangle$, $\langle \texttt{vars}[1], \texttt{vars}[2] \rangle$, ..., $\langle \texttt{vars}[n-2], \texttt{vars}[n-1] \rangle$ 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 $\texttt{vars}[0..n-1]$ 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 `ninf`.

- 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 $\texttt{vars}[0], \ldots, \texttt{vars}[n-1]$. In a second phase the checker computes for each potential value its number of occurrences in the sequence $\texttt{vars}[0], \ldots, \texttt{vars}[n-1]$. 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 $\texttt{vars}[0], \ldots, \texttt{vars}[n-1]$. Its initial state initializes an array of counters to 0. Then it triggers successively a transition for each element $\texttt{vars}[i]$ of the input sequence and increments by 1 the entry corresponding to $\texttt{vars}[i]$. 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 $\texttt{vars}[0], \ldots, \texttt{vars}[n-1]$.

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 $\mathcal{S}_0, \ldots, \mathcal{S}_{m-1}$ 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 $\mathcal{S}_0, \ldots, \mathcal{S}_{m-1}$ outside the automaton. To each subset $\mathcal{S}_i$ of this sequence corresponds a variable S_i with an initial domain ranging over $[min, min + |\mathcal{M}| - 1]$, where min is a fixed integer. To each integer of this range corresponds one of the mutually incompatible conditions of $\mathcal{M}$. The sequences $S_0, \ldots, S_{m-1}$ and $\mathcal{S}_0, \ldots, \mathcal{S}_{m-1}$ 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 the 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

$$\langle \mathcal{S}ignature, \mathcal{S}ignature\mathcal{D}omain, \mathcal{S}ignature\mathcal{A}rg, \mathcal{S}ignature\mathcal{A}rg\mathcal{P}attern, \mathcal{C}ounters, \mathcal{A}rrays, \mathcal{S}tates, \mathcal{T}ransitions \rangle$$

where:

- $\mathcal{S}ignature$ is the sequence of variables $S_0, \ldots, S_{m-1}$ corresponding to the signature of the constraint $\mathcal{C}$.
- $\mathcal{S}ignature\mathcal{D}omain$ is an interval that defines the range of possible values of the variables of $\mathcal{S}ignature$.
- $\mathcal{S}ignature\mathcal{A}rg$ is the signature argument $\mathcal{S}_0, \ldots, \mathcal{S}_{m-1}$ 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, $\mathcal{S}ignature\mathcal{A}rg\mathcal{P}attern$ defines a symbolic name for each term of $\mathcal{S}ignature\mathcal{A}rg$. These names can be used within the description of a transition for expressing an additional condition for triggering the corresponding transition.
- $\mathcal{C}ounters$ is the, possibly empty, list of all counters used in the automaton $\mathcal{A}$. Each counter is described by a term $\mathrm{t}(Counter, InitialValue, FinalVariable)$ 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 $\mathcal{A}$.
- $\mathcal{A}rrays$ is the, possibly empty, list of all arrays used in the automaton $\mathcal{A}$. Each array is described by a term $\mathrm{t}(Array, InitialValue, FinalConstraint)$ 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 $\mathcal{C}ounters$. 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 $\mathcal{A}$.

- $\mathcal{S}tates$ is the list of states of $\mathcal{A}$, where each state has the form $\mathrm{source}(id)$, $\mathrm{sink}(id)$ or $\mathrm{node}(id)$. id is a unique identifier associated with each state. Finally, $\mathrm{source}(id)$ and $\mathrm{sink}(id)$ respectively denote the initial and the final state of $\mathcal{A}$.

- $\mathcal{T}ransitions$ is the list of transitions of $\mathcal{A}$. Each transition t has the form $\mathrm{arc}(id_1, label, id_2)$ or $\mathrm{arc}(id_1, label, id_2, counters)$. 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 $\mathcal{C}ounters$ 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 $\mathcal{C}ounters$.

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

- $\mathcal{S}ignature = S_0, S_1, \ldots, S_{n-2}$,
- $\mathcal{S}ignature\mathcal{D}omain = 0..2$,
- $\mathcal{S}ignature\mathcal{A}rg = \langle vars[0], vars[1]\rangle, \ldots, \langle vars[n-2], vars[n-1]\rangle$,
- $\mathcal{S}ignature\mathcal{A}rg\mathcal{P}attern$ is not used,
- $\mathcal{C}ounters = \mathrm{t}(c, 0, ninf)$,
- $\mathcal{S}tates = [source(s), node(i), node(j), sink(t)]$,
- $\mathcal{T}ransitions = [\mathrm{arc}(s,1,s), \mathrm{arc}(s,2,i), \mathrm{arc}(s,0,j), \mathrm{arc}(s,\$,t), \mathrm{arc}(i,1,i), \mathrm{arc}(i,2,i), \mathrm{arc}(i,0,j,[c+1]), \mathrm{arc}(i,\$,t), \mathrm{arc}(j,1,j), \mathrm{arc}(j,0,j), \mathrm{arc}(j,2,i,[c+1]), \mathrm{arc}(j,\$,t)]$.

The signature constraint relating each pair of variables $\langle vars[i], vars[i+1]\rangle$ to the signature variable S_i is defined as follows: $\Psi_{\texttt{inflexion}}(S_i, vars[i], vars[i+1]) \equiv vars[i] > vars[i+1] \Leftrightarrow S_i = 0 \ \wedge vars[i] = vars[i+1] \Leftrightarrow S_i = 1 \ \wedge vars[i] < vars[i+1] \Leftrightarrow S_i = 2$. The sequence of transitions triggered on the ground instance `inflexion`$(4, [3,3,1,4,5,5,6,5,5,6,3])$ is $\frac{s}{c=0} \xrightarrow{3=3\Leftrightarrow S_0=1} s \xrightarrow{3>1\Leftrightarrow S_1=0} j \xrightarrow[c=1]{1<4\Leftrightarrow S_2=2} i \xrightarrow{4<5\Leftrightarrow S_3=2} i \xrightarrow{5=5\Leftrightarrow S_4=1} i \xrightarrow{5<6\Leftrightarrow S_5=2} i \xrightarrow[c=2]{6>5\Leftrightarrow S_6=0} j \xrightarrow{5=5\Leftrightarrow S_7=1} j \xrightarrow[c=3]{5<6\Leftrightarrow S_8=2} i \xrightarrow[c=4]{6>3\Leftrightarrow S_9=0} j \xrightarrow{\$} \frac{t}{ninf=4}$. 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?

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 mentioned[1]. 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 $\mathbf{NARC}$ graph-property we find the `inverse` and `place_in_pyramid` constraints. Since $\mathbf{NARC}$ is underlined and overlined these constraints will have to be woken each time the minimum or the maximum of $\mathbf{NARC}$ 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 solutions[2]:

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

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

[2] Automata 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&T[3].

[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,
- among_diff_0,
- among_interval,
- among_low_up,
- among_modulo,
- atleast,
- atmost,
- count,
- counts,
- differ_from_at_least_k_pos,
- exactly,
- group,
- group_skip_isolated_item,
- 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,
- group_skip_isolated_item,
- ith_pos_different_from_0.

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.

Apartition:

- 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.

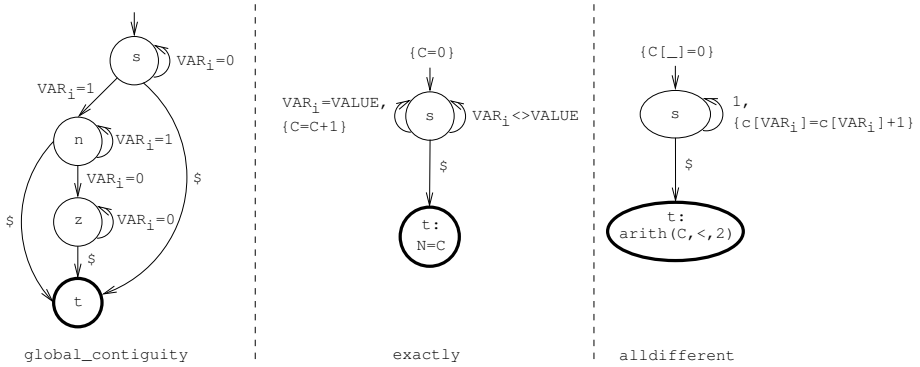


Figure 2.1: Examples of automata

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:

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

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,
- heighest_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.

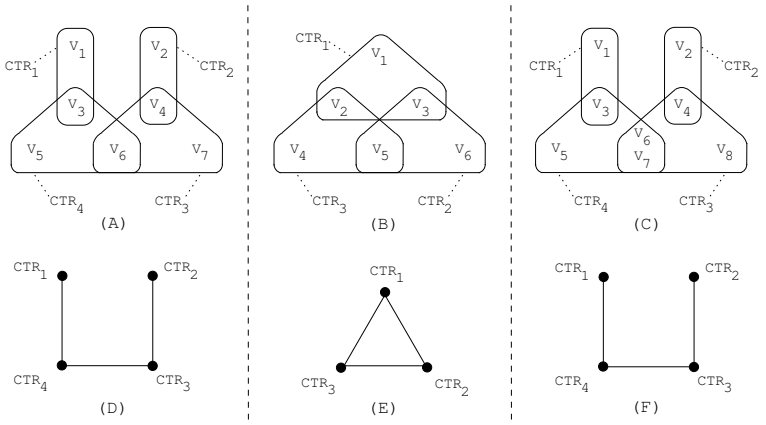


Figure 2.2: Illustration of Berge-acyclic constraint network

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`.

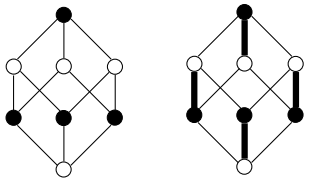


Figure 2.3: A bipartite graph and one of its bipartite matching

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 $B_1, B_2, \ldots, B_n$ 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`.

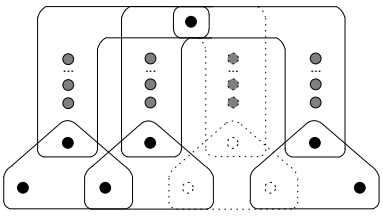


Figure 2.4: Hypergraph associated with a centered cyclic(1) constraint network(1)

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.

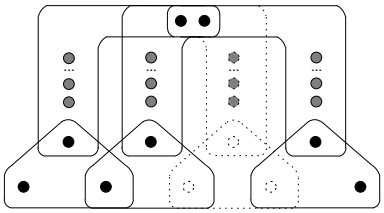


Figure 2.5: Hypergraph associated with a centered cyclic(2) constraint network(1)

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.

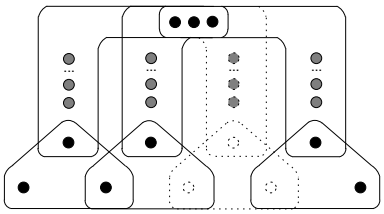


Figure 2.6: Hypergraph associated with a centered cyclic(3) constraint network(1)

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.

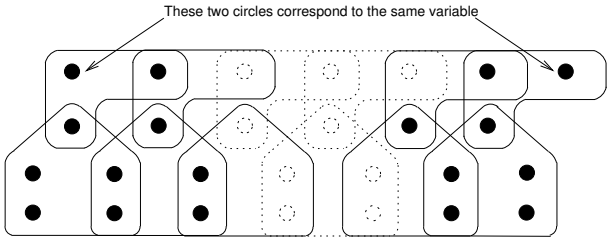


Figure 2.7: Hypergraph corresponding to a circular sliding cyclic(1) constraint network(2)

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.

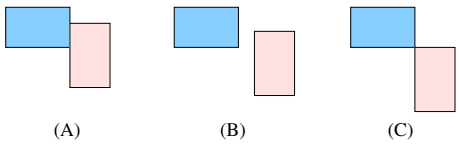


Figure 2.8: Illustration of the notion of contact

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.

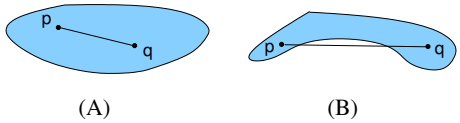


Figure 2.9: A convex set and a non-convex set

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.

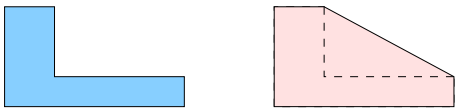


Figure 2.10: Convex hull of a non-convex set

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,
- minimum_weight_alldifferent,
- sum_of_weights_of_distinct_values,
- 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 ($v \in$ Vars, $u \in$ 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 $B_1, B_2, \ldots, B_n$. 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.

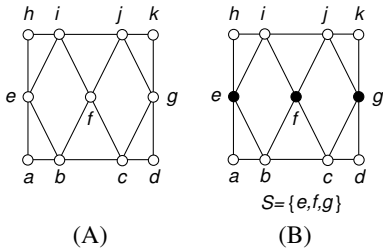


Figure 2.11: A graph and one of its dominating set

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_greater,
- lex_greatereq,
- lex_less,
- lex_lesseq.

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

- 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) $a_1 < \cdots < a_k$ 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 $H = \{v_1, v_2, \ldots, v_h\}$ 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.

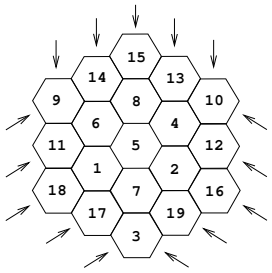


Figure 2.12: A magic hexagon

Magic series:

- `global_cardinality`.

A constraint that allows for modeling the *magic series* problem with one single constraint. A non-empty finite series $S = (s_0, s_1, \ldots, s_n)$ 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,
- all_differ_from_at_least_k_pos,
- among_low_up,
- arith_or,
- cardinality_atleast,
- cardinality_atmost_partition,
- cardinality_atmost,
- change_continuity,
- change_pair,
- change_partition,
- change,
- common_interval,
- common_modulo,
- common_partition,
- common,
- correspondence,
- counts,
- crossing,
- cyclic_change_joker,
- 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 objets 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_between,
- distance_change.

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:

- among_seq,
- arith_sliding,
- cycle_card_on_path,
- deepest_valley,
- heighest_peak,
- inflexion,
- no_peak,
- no_valley,
- peak,
- period,
- period_except_0,
- relaxed_sliding_sum,
- sequence_folding,
- size_maximal_sequence_alldifferent,
- size_maximal_starting_sequence_alldifferent,
- sliding_card_skip0,
- sliding_distribution,
- sliding_sum,
- valley.

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:

- inverse_set,
- link_set_to_booleans.

A channeling constraint involving one or several set variables.

Scheduling constraint:

- coloured_cumulative,
- coloured_cumulatives,
- cumulative,
- cumulative_product,
- cumulative_with_level_of_priority,
- cumulatives,
- disjoint_tasks,
- disjunctive,
- period,
- period_except_0,

- 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.

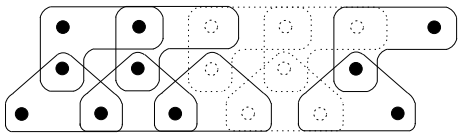


Figure 2.13: Hypergraph associated with a sliding cyclic(1) constraint network(1)

Sliding cyclic(1) constraint network(2):

- change,
- change_continuity,
- cyclic_change,
- cyclic_change_joker,
- deepest_valley,
- heighest_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.

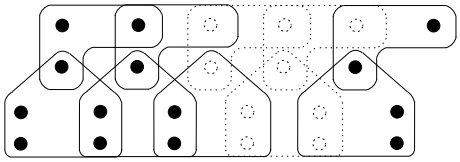


Figure 2.14: Hypergraph associated with a sliding cyclic(1) constraint network(2)

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.

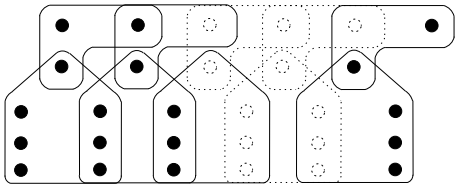


Figure 2.15: Hypergraph associated with a sliding cyclic(1) constraint network(3)

Sliding cyclic(2) constraint network(2):

- `change_pair`,
- `distance_change`.

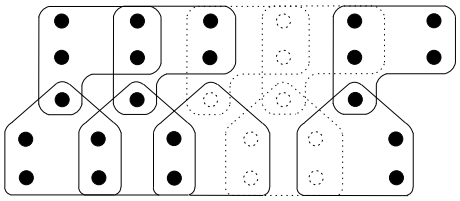


Figure 2.16: Hypergraph associated with a sliding cyclic(2) constraint network(2)

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:

- 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,
- weighted_partial_alldiff.

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_from_0,
- 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_except_0,
- shift,
- sliding_card_skip0,
- smooth,
- stretch_circuit,
- stretch_path,
- symmetric_alldifferent,
- symmetric_cardinality,
- symmetric_gcc,
- track.

A constraint that can occur in timetabling problems.

Time window:

- sliding_time_window_sum.

A constraint involving one or several date ranges.

Touch:

- orths_are_connected,
- two_orth_are_in_contact.

A constraint enforcing that some orthotopes touch each other (see **Contact**).

Tree:

- binary_tree,
- one_tree,
- 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 one tree. Each vertex points to its father or to itself if it corresponds to the root of a tree.

Tuple:

- in_relation,
- vec_eq_tuple.

A constraint involving a *tuple*. A *tuple* is an element of a *relation*, where a *relation* is a subset of the product of several finite sets.

Unary constraint:

- in,
- not_in.

A constraint involving only one variable.

Undirected graph:

- tour.

A constraint that deals with an *undirected graph*. An *undirected graph* is a graph whose edges consist of unordered pairs of vertices.

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,
- minimum_weight_alldifferent,
- sum_of_weights_of_distinct_values,
- 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

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:

  - **MAX_DRG**,
  - **MAX_OD**,
  - **MIN_DRG**,
  - **MIN_ID**,
  - **MIN_OD**,
  - **NTREE**,
  - **PATH_FROM_TO**,
  - **PRODUCT**,
  - **RANGE**,
  - **RANGE_DRG**,
  - **RANGE_NCC**,
  - **SUM**,
  - **SUM_WEIGHT_ARC**.

  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 $\texttt{atleast}_v$ and at most $\texttt{atmost}_v$ 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 $\mathbf{NARC}$ with the number of vertices of the smallest and largest connected component (i.e. $\mathbf{MIN\_NCC}$ and $\mathbf{MAX\_NCC}$).

$$\mathbf{MIN\_NCC} \neq \mathbf{MAX\_NCC} \Rightarrow \mathbf{NARC} \geq \mathbf{MIN\_NCC} + \mathbf{MAX\_NCC} - 2 + (\mathbf{MIN\_NCC} = 1)$$

$$\texttt{equivalence} : \mathbf{MIN\_NCC} \neq \mathbf{MAX\_NCC} \Rightarrow \mathbf{NARC} \geq \mathbf{MIN\_NCC}^2 + \mathbf{MAX\_NCC}^2$$

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 $\mathbf{MIN\_NCC}^2 + \mathbf{MAX\_NCC}^2$ is greater than or equal to $\mathbf{MIN\_NCC} + \mathbf{MAX\_NCC} - 2 + (\mathbf{MIN\_NCC} = 1)$), 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:

$$\mathbf{NARC} \leq \mathbf{NVERTEX}^2$$

$$\mathbf{arc\_gen} = CIRCUIT : \mathbf{NARC} \leq \mathbf{NVERTEX}$$

$$\mathbf{arc\_gen} = CHAIN : \mathbf{NARC} \leq 2 \cdot \mathbf{NVERTEX} - 2$$

$$\mathbf{arc\_gen} = CLIQUE(\leq) : \mathbf{NARC} \leq \frac{\mathbf{NVERTEX} \cdot (\mathbf{NVERTEX} + 1)}{2}$$

$$\mathbf{arc\_gen} = CLIQUE(\geq) : \mathbf{NARC} \leq \frac{\mathbf{NVERTEX} \cdot (\mathbf{NVERTEX} + 1)}{2}$$

$$\mathbf{arc\_gen} = CLIQUE(<) : \mathbf{NARC} \leq \frac{\mathbf{NVERTEX} \cdot (\mathbf{NVERTEX} - 1)}{2}$$

$$\mathbf{arc\_gen} = CLIQUE(>) : \mathbf{NARC} \leq \frac{\mathbf{NVERTEX} \cdot (\mathbf{NVERTEX} - 1)}{2}$$

$$\mathbf{arc\_gen} = CLIQUE(\neq) : \mathbf{NARC} \leq \mathbf{NVERTEX}^2 - \mathbf{NVERTEX}$$

$$\mathbf{arc\_gen} = CYCLE : \mathbf{NARC} \leq 2 \cdot \mathbf{NVERTEX}$$

$$\mathbf{arc\_gen} = PATH : \mathbf{NARC} \leq \mathbf{NVERTEX} - 1$$

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),

  - $\mathbf{MAX\_NCC}$, $\mathbf{MIN\_NCC}$, $\mathbf{NCC}$, $\mathbf{NVERTEX}$: 2 (see Propositions 100 and 101),
  - $\mathbf{MAX\_NSCC}$, $\mathbf{MIN\_NSCC}$, $\mathbf{NARC}$, $\mathbf{NSCC}$: 2 (see Propositions 102 and 103),
  - $\mathbf{MAX\_NSCC}$, $\mathbf{MIN\_NSCC}$, $\mathbf{NSCC}$, $\mathbf{NVERTEX}$: 2 (see Propositions 104 and 105),
  - $\mathbf{MIN\_NCC}$, $\mathbf{NARC}$, $\mathbf{NCC}$, $\mathbf{NVERTEX}$: 1 (see Proposition 106),
  - $\mathbf{NARC}$, $\mathbf{NCC}$, $\mathbf{NSCC}$, $\mathbf{NVERTEX}$: 2 (see Propositions 107 and 108),
  - $\mathbf{NARC}$, $\mathbf{NSINK}$, $\mathbf{NSOURCE}$, $\mathbf{NVERTEX}$: 1 (see Proposition 109).

- Graph invariants involving five graph characteristics of a final graph:
  - $\mathbf{MAX\_NCC}$, $\mathbf{MIN\_NCC}$, $\mathbf{NARC}$, $\mathbf{NCC}$, $\mathbf{NVERTEX}$: 1 (see Proposition 110),
  - $\mathbf{MIN\_NCC}$, $\mathbf{NARC}$, $\mathbf{NCC}$, $\mathbf{NSCC}$, $\mathbf{NVERTEX}$: 1 (see Proposition 111).

- Graph invariants relating two characteristics of two final graphs:
  - $\mathbf{MAX\_NCC}_1$, $\mathbf{NCC}_2$: 1 (see Proposition 112),
  - $\mathbf{MAX\_NCC}_2$, $\mathbf{NCC}_1$: 1 (see Proposition 113),
  - $\mathbf{MIN\_NCC}_1$, $\mathbf{NCC}_2$: 1 (see Proposition 114),
  - $\mathbf{MIN\_NCC}_2$, $\mathbf{NCC}_1$: 1 (see Proposition 115),
  - $\mathbf{NARC}_1$, $\mathbf{NARC}_2$: 1 (see Proposition 116),
  - $\mathbf{NCC}_1$, $\mathbf{NCC}_2$: 2 (see Propositions 117 and 118),
  - $\mathbf{NVERTEX}_1$, $\mathbf{NVERTEX}_2$: 1 (see Proposition 119).

- Graph invariants relating three characteristics of two final graphs:
  - $\mathbf{MAX\_NCC}_1$, $\mathbf{MIN\_NCC}_1$, $\mathbf{MIN\_NCC}_2$: 2 (see Propositions 120 and 121),
  - $\mathbf{MAX\_NCC}_2$, $\mathbf{MIN\_NCC}_2$, $\mathbf{MIN\_NCC}_1$: 2 (see Propositions 122 and 123),
  - $\mathbf{MIN\_NCC}_1$, $\mathbf{NARC}_2$, $\mathbf{NCC}_1$: 1 (see Proposition 124),
  - $\mathbf{MIN\_NCC}_2$, $\mathbf{NARC}_1$, $\mathbf{NCC}_2$: 1 (see Proposition 125).

- Graph invariants relating four characteristics of two final graphs:
  - $\mathbf{MAX\_NCC}_1$, $\mathbf{MIN\_NCC}_1$, $\mathbf{MIN\_NCC}_2$, $\mathbf{NCC}_1$: 2 (see Propositions 126 and 127),
  - $\mathbf{MAX\_NCC}_2$, $\mathbf{MIN\_NCC}_2$, $\mathbf{MIN\_NCC}_1$, $\mathbf{NCC}_2$: 2 (see Propositions 128 and 129).

- Graph invariants relating five characteristics of two final graphs:
  - $\mathbf{MAX\_NCC}_1$, $\mathbf{MAX\_NCC}_2$, $\mathbf{MIN\_NCC}_1$, $\mathbf{MIN\_NCC}_2$, $\mathbf{NCC}_1$: 7 (see Propositions 130, 131, 132, 133, 134, 135 and 136).
  - $\mathbf{MAX\_NCC}_1$, $\mathbf{MAX\_NCC}_2$, $\mathbf{MIN\_NCC}_1$, $\mathbf{MIN\_NCC}_2$, $\mathbf{NCC}_2$: 7 (see Propositions 137, 138, 139, 140, 141, 142 and 143).

- Graph invariants relating six characteristics of two final graphs:
  - $\mathbf{MAX\_NCC}_1$, $\mathbf{MAX\_NCC}_2$, $\mathbf{MIN\_NCC}_1$, $\mathbf{MIN\_NCC}_2$, $\mathbf{NCC}_1$, $\mathbf{NCC}_2$: 2 (see Propositions 144 and 145).

Graph invariants involving one characteristic of a final graph

MAX_NCC

Proposition 1.

$$\texttt{no\_loop} : \mathbf{MAX\_NCC} \neq 1 \tag{3.1}$$

Proof. Since we don't have any loop, a non-empty connected component has at least two vertices. □

MAX_NSCC

Proposition 2.

$$\texttt{acyclic} : \mathbf{MAX\_NSCC} \leq 1 \tag{3.2}$$

Proof. Since we don't have any circuit, a non-empty strongly connected component consists of one single vertex. □

Proposition 3.

$$\texttt{no\_loop} : \mathbf{MAX\_NSCC} \neq 1 \tag{3.3}$$

Proof. Since we don't have any loop, a non-empty strongly connected component has at least two vertices. □

MIN_NCC

Proposition 4.

$$\texttt{no\_loop} : \mathbf{MIN\_NCC} \neq 1 \tag{3.4}$$

Proof. Since we don't have any loop, a non-empty connected component has at least two vertices. □

MIN_NSCC

Proposition 5.

$$\texttt{acyclic} : \mathbf{MIN\_NSCC} \leq 1 \tag{3.5}$$

Proof. Since we don't have any circuit, a non-empty strongly connected component consists of one single vertex. □

Proposition 6.

$$\texttt{no\_loop} : \mathbf{MIN\_NSCC} \neq 1 \tag{3.6}$$

Proof. Since we don't have any loop, a non-empty strongly connected component has at least two vertices. □

NARC

Proposition 7.

$$\texttt{one\_succ} : \mathbf{NARC} = \mathbf{NVERTEX}_{\texttt{INITIAL}} \tag{3.7}$$

Proof. By definition of one_succ. □

NCC

Proposition 8.

$$\texttt{no\_loop} : 2 \cdot \mathbf{NCC} \leq \mathbf{NVERTEX}_{\texttt{INITIAL}} \tag{3.8}$$

Proof. By definition of `no_loop`, each connected component has at least two vertices. □

Proposition 9.

$$\texttt{consecutive\_loops\_are\_connected} : 2 \cdot \mathbf{NCC} \leq \mathbf{NVERTEX}_{\texttt{INITIAL}} + 1 \tag{3.9}$$

Proof. By definition of `consecutive_loops_are_connected`. □

NSCC

Proposition 10.

$$\texttt{no\_loop} : 2 \cdot \mathbf{NSCC} \leq \mathbf{NVERTEX}_{\texttt{INITIAL}} \tag{3.10}$$

Proof. By definition of `no_loop`, each strongly connected component has at least two vertices. □

NSINK

Proposition 11.

$$\texttt{symmetric} : \mathbf{NSINK} = 0 \tag{3.11}$$

Proof. Since we don't have any isolated vertex. □

NSOURCE

Proposition 12.

$$\texttt{symmetric} : \mathbf{NSOURCE} = 0 \tag{3.12}$$

Proof. Since we don't have any isolated vertex. □

NVERTEX

Proposition 13.

$$\texttt{one\_succ} : \mathbf{NVERTEX} = \mathbf{NVERTEX}_{\texttt{INITIAL}} \tag{3.13}$$

Proof. By definition of `one_succ`. □

Graph invariants involving two characteristics of a final graph

MAX_NCC, MAX_NSCC

Proposition 14.

$$\mathbf{MAX\_NCC} = 0 \Leftrightarrow \mathbf{MAX\_NSCC} = 0 \tag{3.14}$$

Proof. By definition of $\mathbf{MAX\_NCC}$ and of $\mathbf{MAX\_NSCC}$. □

Proposition 15.

$$\mathbf{MAX\_NSCC} \leq \mathbf{MAX\_NCC} \tag{3.15}$$

Proof. $\mathbf{MAX\_NSCC}$ 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.

$$\mathbf{MAX\_NCC} = 0 \Leftrightarrow \mathbf{MIN\_NCC} = 0 \tag{3.16}$$

Proof. By definition of $\mathbf{MAX\_NCC}$ and of $\mathbf{MIN\_NCC}$. □

Proposition 17.

$$\mathbf{MIN\_NCC} \leq \mathbf{MAX\_NCC} \tag{3.17}$$

Proof. By definition of $\mathbf{MIN\_NCC}$ and of $\mathbf{MAX\_NCC}$. □

MAX_NCC, NARC

Proposition 18.

$$\mathbf{MAX\_NCC} = 0 \Leftrightarrow \mathbf{NARC} = 0 \tag{3.18}$$

Proof. By definition of $\mathbf{MAX\_NCC}$ and of $\mathbf{NARC}$. □

Proposition 19.

$$\mathbf{MAX\_NCC} > 0 \Rightarrow \mathbf{NARC} \geq \max(1, \mathbf{MAX\_NCC} - 1) \tag{3.19}$$

$$\texttt{symmetric} : \mathbf{MAX\_NCC} > 0 \Rightarrow \mathbf{NARC} \geq \max(1, 2 \cdot \mathbf{MAX\_NCC} - 2) \tag{3.20}$$

$$\texttt{equivalence} : \mathbf{NARC} \geq \mathbf{MAX\_NCC}^2 \tag{3.21}$$

$$\mathbf{arc\_gen} = \mathit{PATH} : \mathbf{NARC} \geq \mathbf{MAX\_NCC} - 1 \tag{3.22}$$

Proof.
(3.19) $\mathbf{MAX\_NCC}-1$ arcs are needed to connect $\mathbf{MAX\_NCC}$ 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, $2 \cdot \mathbf{MAX\_NCC} - 2$ arcs are needed to connect $\mathbf{MAX\_NCC}$ vertices that belong to a given connected component containing at least two vertices.

(3.21) Finally, when the graph is reflexive, symmetric and transitive, $\mathbf{MAX\_NCC}^2$ arcs are needed to connect $\mathbf{MAX\_NCC}$ 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.

$$\mathbf{MAX\_NCC} = 0 \Rightarrow \mathbf{NSINK} = 0 \tag{3.23}$$

Proof. By definition of $\mathbf{MAX\_NCC}$ and of $\mathbf{NSINK}$. □

Proposition 21.

$$\mathbf{NSINK} \geq 1 \Rightarrow \mathbf{MAX\_NCC} \geq 2 \tag{3.24}$$

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.

$$\mathbf{MAX\_NCC} = 0 \Rightarrow \mathbf{NSOURCE} = 0 \tag{3.25}$$

Proof. By definition of $\mathbf{MAX\_NCC}$ and of $\mathbf{NSOURCE}$. □

Proposition 23.

$$\mathbf{NSOURCE} \geq 1 \Rightarrow \mathbf{MAX\_NCC} \geq 2 \tag{3.26}$$

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.

$$\mathbf{MAX\_NCC} = 0 \Leftrightarrow \mathbf{NVERTEX} = 0 \tag{3.27}$$

Proof. By definition of $\mathbf{MAX\_NCC}$ and of $\mathbf{NVERTEX}$. □

Proposition 25.

$$\mathbf{NVERTEX} \geq \mathbf{MAX\_NCC} \tag{3.28}$$

Proof. By definition of $\mathbf{MAX\_NCC}$. □

MAX_NSCC, MIN_NSCC

Proposition 26.

$$\mathbf{MAX\_NSCC} = 0 \Leftrightarrow \mathbf{MIN\_NSCC} = 0 \tag{3.29}$$

Proof. By definition of $\mathbf{MAX\_NSCC}$ and of $\mathbf{MIN\_NSCC}$. □

Proposition 27.

$$\mathbf{MIN\_NSCC} \leq \mathbf{MAX\_NSCC} \tag{3.30}$$

Proof. By definition of $\mathbf{MIN\_NSCC}$ and of $\mathbf{MAX\_NSCC}$. □

MAX_NSCC, NARC

Proposition 28.

$$\mathbf{MAX\_NSCC} = 0 \Leftrightarrow \mathbf{NARC} = 0 \tag{3.31}$$

Proof. By definition of $\mathbf{MAX\_NSCC}$ and of $\mathbf{NARC}$. □

Proposition 29.

$$\mathbf{NARC} \geq \mathbf{MAX\_NSCC} \tag{3.32}$$

$$\texttt{symmetric} : \mathbf{NARC} \geq 2 \cdot \mathbf{MAX\_NSCC} \tag{3.33}$$

$$\texttt{equivalence} : \mathbf{NARC} \geq \mathbf{MAX\_NSCC}^2 \tag{3.34}$$

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 $\mathbf{MAX\_NSCC}$ vertices this leads to the previous inequality. □

MAX_NSCC, NVERTEX

Proposition 30.

$$\mathbf{MAX\_NSCC} = 0 \Leftrightarrow \mathbf{NVERTEX} = 0 \tag{3.35}$$

Proof. By definition of $\mathbf{MAX\_NSCC}$ and of $\mathbf{NVERTEX}$. □

Proposition 31.

$$\mathbf{NVERTEX} \geq \mathbf{MAX\_NSCC} \tag{3.36}$$

Proof. By definition of $\mathbf{MAX\_NSCC}$. □

MIN_NCC, MIN_NSCC

Proposition 32.

$$\mathbf{MIN\_NCC} = 0 \Leftrightarrow \mathbf{MIN\_NSCC} = 0 \tag{3.37}$$

Proof. By definition of $\mathbf{MIN\_NCC}$ and of $\mathbf{MIN\_NSCC}$. □

Proposition 33.

$$\mathbf{MIN\_NCC} \geq \mathbf{MIN\_NSCC} \tag{3.38}$$

Proof. By construction $\mathbf{MIN\_NCC}$ is an upper bound of the number of vertices of the smallest strongly connected component. □

MIN_NCC, NARC

Proposition 34.

$$\mathbf{MIN\_NCC} = 0 \Leftrightarrow \mathbf{NARC} = 0 \tag{3.39}$$

Proof. By definition of $\mathbf{MIN\_NCC}$ and of $\mathbf{NARC}$. □

Proposition 35.

$$\mathbf{MIN\_NCC} > 0 \Rightarrow \mathbf{NARC} \geq \max(1, \mathbf{MIN\_NCC} - 1) \tag{3.40}$$

$$\texttt{symmetric} : \mathbf{MIN\_NCC} > 0 \Rightarrow \mathbf{NARC} \geq \max(1, 2 \cdot \mathbf{MIN\_NCC} - 2) \tag{3.41}$$

$$\texttt{equivalence} : \mathbf{NARC} \geq \mathbf{MIN\_NCC}^2 \tag{3.42}$$

$$\mathbf{arc\_gen} = PATH : \mathbf{NARC} \geq \mathbf{MIN\_NCC} - 1 \tag{3.43}$$

Proof. Similar to Proposition 19. □

MIN_NCC, NCC

Proposition 36.

$$\texttt{consecutive\_loops\_are\_connected} : (\mathbf{MIN\_NCC}+1) \cdot \mathbf{NCC} \leq \mathbf{NVERTEX}_{\texttt{INITIAL}} + 1 \tag{3.44}$$

Proof. By definition of `consecutive_loops_are_connected`. □

MIN_NCC, NVERTEX

Proposition 37.

$$\mathbf{MIN\_NCC} = 0 \Leftrightarrow \mathbf{NVERTEX} = 0 \tag{3.45}$$

Proof. By definition of $\mathbf{MIN\_NCC}$ and of $\mathbf{NVERTEX}$. □

Proposition 38.

$$\mathbf{NVERTEX} \geq \mathbf{MIN\_NCC} \tag{3.46}$$

Proof. By definition of $\mathbf{MIN\_NCC}$. □

Proposition 39.

$$\mathbf{MIN\_NCC} \notin \left[\min\left(\left\lfloor \frac{\mathbf{NVERTEX}}{2} \right\rfloor, \left\lfloor \frac{\mathbf{NVERTEX}_{\texttt{INITIAL}} - 1}{2} \right\rfloor\right) + 1, \mathbf{NVERTEX} - 1\right] \tag{3.47}$$

Proof. On the one hand, if $\mathbf{NCC} \leq 1$, we have that $\mathbf{MIN\_NCC} \geq \mathbf{NVERTEX}$. On the other hand, if $\mathbf{NCC} > 1$, we have that $\mathbf{MIN\_NCC} + \mathbf{MIN\_NCC} \leq \mathbf{NVERTEX}$ and that $\mathbf{MIN\_NCC} + \mathbf{MIN\_NCC} + 1 \leq \mathbf{NVERTEX}_{\texttt{INITIAL}}$, which by isolating $\mathbf{MIN\_NCC}$ and by grouping the two inequalities leads to $\mathbf{MIN\_NCC} \leq \min\left(\left\lfloor \frac{\mathbf{NVERTEX}}{2} \right\rfloor, \left\lfloor \frac{\mathbf{NVERTEX}_{\texttt{INITIAL}}-1}{2} \right\rfloor\right)$. The result follows. □

MIN_NSCC, NARC

Proposition 40.

$$\mathbf{MIN\_NSCC} = 0 \Leftrightarrow \mathbf{NARC} = 0 \tag{3.48}$$

Proof. By definition of **MIN_NSCC** and of **NARC**. □

Proposition 41.

$$\mathbf{NARC} \geq \mathbf{MIN\_NSCC} \tag{3.49}$$

$$\texttt{symmetric} : \mathbf{NARC} \geq 2 \cdot \mathbf{MIN\_NSCC} \tag{3.50}$$

$$\texttt{equivalence} : \mathbf{NARC} \geq \mathbf{MIN\_NSCC}^2 \tag{3.51}$$

Proof. Similar to Proposition 29. □

MIN_NSCC, NVERTEX

Proposition 42.

$$\mathbf{MIN\_NSCC} = 0 \Leftrightarrow \mathbf{NVERTEX} = 0 \tag{3.52}$$

Proof. By definition of **MIN_NSCC** and of **NVERTEX**. □

Proposition 43.

$$\mathbf{NVERTEX} \geq \mathbf{MIN\_NSCC} \tag{3.53}$$

Proof. By definition of **MIN_NSCC**. □

NARC, NCC

Proposition 44.

$$\mathbf{NARC} = 0 \Leftrightarrow \mathbf{NCC} = 0 \tag{3.54}$$

Proof. By definition of **NARC** and of **NCC**. □

Proposition 45.

$$\mathbf{NARC} \geq \mathbf{NCC} \tag{3.55}$$

Proof. Each connected component contains at least one arc (since, by hypothesis, each vertex has at least one arc). □

NARC, NSCC

Proposition 46.

$$\mathbf{NARC} = 0 \Leftrightarrow \mathbf{NSCC} = 0 \tag{3.56}$$

Proof. By definition of **NARC** and of **NSCC**. □

Proposition 47.

$$\mathbf{NARC} \geq \mathbf{NSCC} \tag{3.57}$$

$$\texttt{no\_loop} : \mathbf{NARC} \geq 2 \cdot \mathbf{NSCC} \tag{3.58}$$

Proof. 3.57 (respectively 3.58) holds since each strongly connected component contains at least one (respectively two) arc(s). □

NARC, NVERTEX

Proposition 48.

$$\mathbf{NARC} = 0 \Leftrightarrow \mathbf{NVERTEX} = 0 \tag{3.59}$$

Proof. By definition of $\mathbf{NARC}$ and of $\mathbf{NVERTEX}$. □

Proposition 49.

$$\mathbf{NARC} \leq \mathbf{NVERTEX}^2 \tag{3.60}$$

$$\mathbf{arc\_gen} = CIRCUIT : \mathbf{NARC} \leq \mathbf{NVERTEX} \tag{3.61}$$

$$\mathbf{arc\_gen} = CHAIN : \mathbf{NARC} \leq 2 \cdot \mathbf{NVERTEX} - 2 \tag{3.62}$$

$$\mathbf{arc\_gen} = CLIQUE(\leq) : \mathbf{NARC} \leq \frac{\mathbf{NVERTEX} \cdot (\mathbf{NVERTEX} + 1)}{2} \tag{3.63}$$

$$\mathbf{arc\_gen} = CLIQUE(\geq) : \mathbf{NARC} \leq \frac{\mathbf{NVERTEX} \cdot (\mathbf{NVERTEX} + 1)}{2} \tag{3.64}$$

$$\mathbf{arc\_gen} = CLIQUE(<) : \mathbf{NARC} \leq \frac{\mathbf{NVERTEX} \cdot (\mathbf{NVERTEX} - 1)}{2} \tag{3.65}$$

$$\mathbf{arc\_gen} = CLIQUE(>) : \mathbf{NARC} \leq \frac{\mathbf{NVERTEX} \cdot (\mathbf{NVERTEX} - 1)}{2} \tag{3.66}$$

$$\mathbf{arc\_gen} = CLIQUE(\neq) : \mathbf{NARC} \leq \mathbf{NVERTEX}^2 - \mathbf{NVERTEX} \tag{3.67}$$

$$\mathbf{arc\_gen} = CYCLE : \mathbf{NARC} \leq 2 \cdot \mathbf{NVERTEX} \tag{3.68}$$

$$\mathbf{arc\_gen} = PATH : \mathbf{NARC} \leq \mathbf{NVERTEX} - 1 \tag{3.69}$$

Proof. 3.60 holds since each vertex of a digraph can have at most $\mathbf{NVERTEX}$ successors. The next items correspond to the maximum number of arcs that can be achieved according to a specific arc generator. □

Proposition 50.

$$2 \cdot \mathbf{NARC} \geq \mathbf{NVERTEX} \tag{3.70}$$

Proof. By induction on the number of vertices of a graph G:

1. If $\mathbf{NVERTEX}(G)$ is equal to 1 or 2 Proposition 50 holds.
2. Assume that $\mathbf{NVERTEX}(G) \geq 3$.
   - 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 $\mathbf{NARC}(G) \geq \mathbf{NARC}(G - v) + 1$. Thus $2 \cdot \mathbf{NARC}(G) \geq 2 \cdot \mathbf{NARC}(G - v) + 1$. Since by induction hypothesis $2 \cdot \mathbf{NARC}(G - v) \geq \mathbf{NVERTEX}(G - v) = \mathbf{NVERTEX}(G) - 1$ 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 $\mathbf{NARC}(G) = \mathbf{NARC}(G - \{v, w\}) + 1$. As by induction hypothesis, $2 \cdot \mathbf{NARC}(G - \{v, w\}) \geq \mathbf{NVERTEX}(G - \{v, w\}) = \mathbf{NVERTEX}(G) - 2$ the result holds.

□

Proposition 51.

$$\mathbf{arc\_gen} = LOOP : \mathbf{NARC} = \mathbf{NVERTEX} \tag{3.71}$$

Proof. From the definition of $LOOP$. □

NCC, NSCC

Proposition 52.

$$\mathbf{NCC} = 0 \Leftrightarrow \mathbf{NSCC} = 0 \tag{3.72}$$

Proof. By definition of $\mathbf{NCC}$ and of $\mathbf{NSCC}$. □

Proposition 53.

$$\mathbf{NCC} \leq \mathbf{NSCC} \tag{3.73}$$

Proof. Holds since each connected component contains at least one strongly connected component. □

NCC, NVERTEX

Proposition 54.

$$\mathbf{NCC} = 0 \Leftrightarrow \mathbf{NVERTEX} = 0 \tag{3.74}$$

Proof. By definition of $\mathbf{NCC}$ and of $\mathbf{NVERTEX}$. □

Proposition 55.

$$\mathbf{NCC} \leq \mathbf{NVERTEX} \tag{3.75}$$

$$\texttt{no\_loop} : 2 \cdot \mathbf{NCC} \leq \mathbf{NVERTEX} \tag{3.76}$$

Proof. 3.75 (respectively 3.76) holds since each connected component contains at least one (respectively two) vertex. □

Proposition 56.

$$\begin{gathered}\texttt{vpartition} \wedge \texttt{consecutive\_loops\_are\_connected} : \\ \mathbf{NVERTEX} \leq \mathbf{NVERTEX}_{\texttt{INITIAL}} - (\mathbf{NCC} - 1)\end{gathered} \tag{3.77}$$

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.

$$\mathbf{NSCC} = 0 \Leftrightarrow \mathbf{NVERTEX} = 0 \tag{3.78}$$

Proof. By definition of $\mathbf{NSCC}$ and of $\mathbf{NVERTEX}$. □

Proposition 58.

$$\mathbf{NSCC} \leq \mathbf{NVERTEX} \tag{3.79}$$

$$\texttt{no\_loop} : 2 \cdot \mathbf{NSCC} \leq \mathbf{NVERTEX} \tag{3.80}$$

Proof. 3.79 (respectively 3.80) holds since each strongly connected component contains at least one (respectively 2) vertex. □

Proposition 59.

$$\texttt{acyclic} : \mathbf{NSCC} = \mathbf{NVERTEX} \tag{3.81}$$

Proof. In a directed acyclic graph we have that each vertex corresponds to a strongly connected component involving only that vertex. □

NSINK, NVERTEX

Proposition 60.

$$\mathbf{NVERTEX} = 0 \Rightarrow \mathbf{NSINK} = 0 \tag{3.82}$$

Proof. By definition of $\mathbf{NVERTEX}$ and of $\mathbf{NSINK}$. □

Proposition 61.

$$\mathbf{NVERTEX} > 0 \Rightarrow \mathbf{NSINK} < \mathbf{NVERTEX} \tag{3.83}$$

Proof. Holds since each sink must have a predecessor which cannot be a sink and since each vertex has at least one arc. □

NSOURCE, NVERTEX

Proposition 62.

$$\mathbf{NVERTEX} = 0 \Rightarrow \mathbf{NSOURCE} = 0 \tag{3.84}$$

Proof. By definition of $\mathbf{NVERTEX}$ and of $\mathbf{NSOURCE}$. □

Proposition 63.

$$\mathbf{NVERTEX} > 0 \Rightarrow \mathbf{NSOURCE} < \mathbf{NVERTEX} \tag{3.85}$$

Proof. Holds since each source must have a successor which cannot be a source and since each vertex has at least one arc. □

Graph invariants involving three characteristics of a final graph

$\boxed{\mathbf{MAX\_NCC}, \mathbf{MIN\_NCC}, \mathbf{NARC}}$

Proposition 64.

$$\begin{aligned}&\mathbf{MIN\_NCC} \neq \mathbf{MAX\_NCC} \Rightarrow \\ &\mathbf{NARC} \geq \mathbf{MIN\_NCC} + \mathbf{MAX\_NCC} - 2 + (\mathbf{MIN\_NCC} = 1)\end{aligned} \tag{3.86}$$

$$\begin{aligned}&\texttt{equivalence} : \mathbf{MIN\_NCC} \neq \mathbf{MAX\_NCC} \Rightarrow \\ &\mathbf{NARC} \geq \mathbf{MIN\_NCC}^2 + \mathbf{MAX\_NCC}^2\end{aligned} \tag{3.87}$$

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 $\mathbf{MIN\_NCC}$ and $\mathbf{MAX\_NCC}$ vertices this leads to the previous inequality. When $\mathbf{MIN\_NCC}$ is equal to one we need an extra arc. □

$\boxed{\mathbf{MAX\_NCC}, \mathbf{MIN\_NCC}, \mathbf{NCC}}$

Proposition 65.

$$\mathbf{MIN\_NCC} \neq \mathbf{MAX\_NCC} \Rightarrow \mathbf{NCC} \geq 2 \tag{3.88}$$

Proof. If $\mathbf{MIN\_NCC}$ and $\mathbf{MAX\_NCC}$ are different then they correspond for sure to at least two distinct connected components. □

$\boxed{\mathbf{MAX\_NCC}, \mathbf{MIN\_NCC}, \mathbf{NVERTEX}}$

Proposition 66.

$$\mathbf{MIN\_NCC} \neq \mathbf{MAX\_NCC} \Rightarrow \mathbf{NVERTEX} \geq \mathbf{MIN\_NCC} + \mathbf{MAX\_NCC} \tag{3.89}$$

Proof. Since we have at least two distinct connected components which respectively have $\mathbf{MIN\_NCC}$ and $\mathbf{MAX\_NCC}$ vertices this leads to the previous inequality. □

Proposition 67.

$$\mathbf{MAX\_NCC} \leq \max(\mathbf{MIN\_NCC}, \mathbf{NVERTEX} - \max(1, \mathbf{MIN\_NCC})) \tag{3.90}$$

Proof. On the one hand, if $\mathbf{NCC} \leq 1$, we have that $\mathbf{MAX\_NCC} \leq \mathbf{MIN\_NCC}$. On the other hand, if $\mathbf{NCC} > 1$, we have that $\mathbf{NVERTEX} \geq \max(1, \mathbf{MIN\_NCC}) + \mathbf{MAX\_NCC}$ (i.e. $\mathbf{MAX\_NCC} \leq \mathbf{NVERTEX} - \max(1, \mathbf{MIN\_NCC})$). The result is obtained by taking the maximum value of the right hand side of the two inequalities. □

Proposition 68.

$$\mathbf{MIN\_NCC} \notin [\mathbf{NVERTEX} - \max(1, \mathbf{MAX\_NCC}) + 1, \mathbf{NVERTEX} - 1] \tag{3.91}$$

Proof. On the one hand, if $\mathbf{NCC} \leq 1$, we have that $\mathbf{MIN\_NCC} \geq \mathbf{NVERTEX}$. On the other hand, if $\mathbf{NCC} > 1$, we have that $\mathbf{MIN\_NCC} + \max(1, \mathbf{MAX\_NCC}) \leq \mathbf{NVERTEX}$ (i.e. $\mathbf{MIN\_NCC} \leq \mathbf{NVERTEX} - \max(1, \mathbf{MAX\_NCC})$). The result follows. □

Proposition 69.

$$\mathbf{NVERTEX} \notin [\mathbf{MIN\_NCC} + 1, \mathbf{MIN\_NCC} + \mathbf{MAX\_NCC} - 1] \tag{3.92}$$

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. □

Proposition 70.

$$\texttt{if } \overline{\mathbf{MIN\_NCC}} > 0$$
$$\texttt{then } k_{inf} = \left\lfloor \frac{\mathbf{NVERTEX} + \overline{\mathbf{MIN\_NCC}}}{\overline{\mathbf{MIN\_NCC}}} \right\rfloor \texttt{ else } k_{inf} = 1$$

$$\texttt{if } \underline{\mathbf{MAX\_NCC}} > 0$$
$$\texttt{then } k_{sup_1} = \left\lfloor \frac{\overline{\mathbf{NVERTEX}} - 1}{\underline{\mathbf{MAX\_NCC}}} \right\rfloor \texttt{ else } k_{sup_1} = \overline{\mathbf{NVERTEX}}$$

$$\texttt{if } \underline{\mathbf{MAX\_NCC}} < \overline{\mathbf{MIN\_NCC}}$$
$$\texttt{then } k_{sup_2} = \left\lfloor \frac{\overline{\mathbf{MIN\_NCC}} - 2}{\underline{\mathbf{MAX\_NCC}} - \overline{\mathbf{MIN\_NCC}}} \right\rfloor \texttt{ else } k_{sup_2} = \overline{\mathbf{NVERTEX}}$$

$$k_{sup} = \min(k_{sup_1}, k_{sup_2})$$

$$\forall k \in [k_{inf}, k_{sup}] : \mathbf{NVERTEX} \notin [k \cdot \mathbf{MAX\_NCC} + 1, (k+1) \cdot \mathbf{MIN\_NCC} - 1] \tag{3.93}$$

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 $[k \cdot \mathbf{MAX\_NCC} + 1, (k+1) \cdot \mathbf{MIN\_NCC} - 1]$. □

MAX_NCC, NARC, NCC

Proposition 71.

$$\mathbf{NARC} \leq \mathbf{NCC} \cdot \mathbf{MAX\_NCC}^2 \tag{3.94}$$

$$\mathbf{arc\_gen} = PATH : \mathbf{NARC} \leq \mathbf{NCC} \cdot (\mathbf{MAX\_NCC} - 1) \tag{3.95}$$

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. □

Proposition 72.

$$\mathbf{NARC} \geq \mathbf{MAX\_NCC} + \mathbf{NCC} - 2 \tag{3.96}$$

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. □

$\mathbf{MAX\_NCC}, \mathbf{NARC}, \mathbf{NVERTEX}$

Proposition 73.

$$\mathbf{MAX\_NCC} > 0 \Rightarrow$$
$$\mathbf{NARC} \leq \mathbf{MAX\_NCC}^2 \cdot \left\lfloor \frac{\mathbf{NVERTEX}}{\mathbf{MAX\_NCC}} \right\rfloor + (\mathbf{NVERTEX} \bmod \mathbf{MAX\_NCC})^2 \tag{3.97}$$

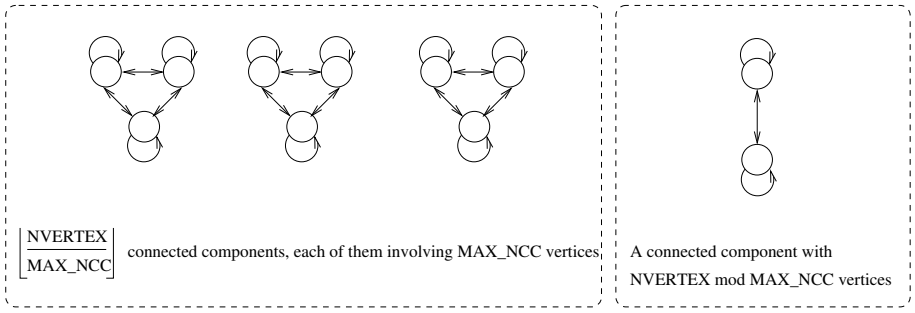


Figure 3.1: Illustration of Proposition 73. A graph that achieves the maximum number of arcs according to the size of the largest connected component as well as to a fixed number of vertices ($\mathbf{MAX\_NCC} = 3, \mathbf{NVERTEX} = 11, \mathbf{NARC} = 3^2 \cdot \lfloor \frac{11}{3} \rfloor + (11 \bmod 3)^2 = 31$)

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 $r(G) = \left\lfloor \frac{\mathbf{NVERTEX}(G)}{\mathbf{MAX\_NCC}(G)} \right\rfloor - \mathbf{NCC}(G, \mathbf{MAX\_NCC}(G))$, where G is any graph. For $r(G) = 0$ the result holds (see Prop 44). Otherwise, since $r(G) > 0$ we have that $V(G) - \mathbf{NCC}(G, \mathbf{MAX\_NCC}(G)) * \mathbf{MAX\_NCC}(G) \geq \mathbf{MAX\_NCC}(G)$, 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. □

Proposition 74.

$$\mathbf{NARC} \geq \mathbf{MAX\_NCC} - 1 + \left\lfloor \frac{\mathbf{NVERTEX} - \mathbf{MAX\_NCC} + 1}{2} \right\rfloor \tag{3.98}$$

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 $\mathbf{NCC} = 1$ in 3.134 of Proposition 89, we have $|E(G[X]) \geq |X| - 1$, on the other hand, by Proposition 50, $|E(G[V(G) - X])| \geq \left\lceil \frac{|V(G) - X|}{2} \right\rceil$. Thus the result follows. □

MAX_NCC, NCC, NVERTEX

Proposition 75.

$$\mathbf{NVERTEX} \leq \mathbf{NCC} \cdot \mathbf{MAX\_NCC} \tag{3.99}$$

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. □

Proposition 76.

$$\mathbf{NVERTEX} \geq \mathbf{MAX\_NCC} + \max(0, \mathbf{NCC} - 1) \tag{3.100}$$

$$\texttt{no\_loop} : \mathbf{NVERTEX} \geq \mathbf{MAX\_NCC} + \max(0, 2 \cdot \mathbf{NCC} - 2) \tag{3.101}$$

Proof. (3.100) The minimum number of vertices according to a fixed number of connected components $\mathbf{NCC}$ such that one of the connected component contains $\mathbf{MAX\_NCC}$ vertices is obtained as follows: We get $\mathbf{MAX\_NCC}$ vertices from the connected component involving $\mathbf{MAX\_NCC}$ vertices and one vertex for each remaining connected component. □

MAX_NSCC, MIN_NSCC, NARC

Proposition 77.

$$\mathbf{MIN\_NSCC} \neq \mathbf{MAX\_NSCC} \Rightarrow \mathbf{NARC} \geq \mathbf{MIN\_NSCC} + \mathbf{MAX\_NSCC} \tag{3.102}$$

$$\begin{aligned} \texttt{equivalence} : \mathbf{MIN\_NSCC} \neq \mathbf{MAX\_NSCC} \Rightarrow \\ \mathbf{NARC} \geq \mathbf{MIN\_NSCC}^2 + \mathbf{MAX\_NSCC}^2 \end{aligned} \tag{3.103}$$

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 $\mathbf{MIN\_NSCC}$ and $\mathbf{MAX\_NSCC}$ vertices this leads to the previous inequality. □

MAX_NSCC, MIN_NSCC, NSCC

Proposition 78.

$$\mathbf{MIN\_NSCC} \neq \mathbf{MAX\_NSCC} \Rightarrow \mathbf{NSCC} \geq 2 \tag{3.104}$$

Proof. Follows from the definitions of $\mathbf{MIN\_NSCC}$ and of $\mathbf{MAX\_NSCC}$. □

MAX_NSCC, MIN_NSCC, NVERTEX

Proposition 79.

$$\mathbf{MIN\_NSCC} \neq \mathbf{MAX\_NSCC} \Rightarrow \mathbf{NVERTEX} \geq \mathbf{MIN\_NSCC} + \mathbf{MAX\_NSCC} \tag{3.105}$$

Proof. Since we have at least two distinct strongly connected components which respectively have $\mathbf{MIN\_NSCC}$ and $\mathbf{MAX\_NSCC}$ vertices this leads to the previous inequality. □

Proposition 80.

$$\texttt{if } \overline{\mathbf{MIN\_NSCC}} > 0$$
$$\texttt{then } k_{inf} = \left\lfloor \frac{\mathbf{NVERTEX} + \overline{\mathbf{MIN\_NSCC}}}{\overline{\mathbf{MIN\_NSCC}}} \right\rfloor \texttt{ else } k_{inf} = 1$$

$$\texttt{if } \underline{\mathbf{MAX\_NSCC}} > 0$$
$$\texttt{then } k_{sup_1} = \left\lfloor \frac{\overline{\mathbf{NVERTEX}} - 1}{\underline{\mathbf{MAX\_NSCC}}} \right\rfloor \texttt{ else } k_{sup_1} = \overline{\mathbf{NVERTEX}}$$

$$\texttt{if } \underline{\mathbf{MAX\_NSCC}} < \overline{\mathbf{MIN\_NSCC}}$$
$$\texttt{then } k_{sup_2} = \left\lfloor \frac{\overline{\mathbf{MIN\_NSCC}} - 2}{\underline{\mathbf{MAX\_NSCC}} - \overline{\mathbf{MIN\_NSCC}}} \right\rfloor \texttt{ else } k_{sup_2} = \overline{\mathbf{NVERTEX}}$$

$$k_{sup} = \min(k_{sup_1}, k_{sup_2})$$

$$\forall k \in [k_{inf}, k_{sup}] : \mathbf{NVERTEX} \notin [k \cdot \mathbf{MAX\_NSCC} + 1, (k+1) \cdot \mathbf{MIN\_NSCC} - 1] \tag{3.106}$$

Proof. Similar to Proposition 70. □

MAX_NSCC, NSCC, NVERTEX

Proposition 81.

$$\mathbf{NVERTEX} \leq \mathbf{NSCC} \cdot \mathbf{MAX\_NSCC} \tag{3.107}$$

Proof. Since each strongly connected component contains at most $\mathbf{MAX\_NSCC}$ vertices the total number of vertices is less than or equal to $\mathbf{NSCC} \cdot \mathbf{MAX\_NSCC}$. □

Proposition 82.

$$\mathbf{NVERTEX} \geq \mathbf{MAX\_NSCC} + \max(0, \mathbf{NSCC} - 1) \tag{3.108}$$

$$\texttt{no\_loop} : \mathbf{NVERTEX} \geq \mathbf{MAX\_NSCC} + \max(0, 2 \cdot \mathbf{NSCC} - 2) \tag{3.109}$$

Proof. (3.108) The minimum number of vertices according to a fixed number of strongly connected components $\mathbf{NSCC}$ such that one of them contains $\mathbf{MAX\_NSCC}$ vertices is equal to $\mathbf{MAX\_NSCC} + \max(0, \mathbf{NSCC} - 1)$. □

MIN_NCC, NARC, NVERTEX

Proposition 83.

$$\mathbf{NARC} \leq \mathbf{MIN\_NCC}^2 + (\mathbf{NVERTEX} - \mathbf{MIN\_NCC})^2 \tag{3.110}$$

$$\mathbf{arc\_gen} = \mathit{CIRCUIT} : \mathbf{NARC} \leq \mathbf{NVERTEX} - 2 \cdot (\mathbf{MIN\_NCC} < \mathbf{NVERTEX}) \tag{3.111}$$

$$\mathbf{arc\_gen} = \mathit{CHAIN} : \mathbf{NARC} \leq \mathbf{NVERTEX} - 2 \cdot (\mathbf{MIN\_NCC} < \mathbf{NVERTEX}) \tag{3.112}$$

$$\mathbf{arc\_gen} = \mathit{CLIQUE}(\leq) : \mathbf{NARC} \leq \frac{\mathbf{MIN\_NCC} \cdot (\mathbf{MIN\_NCC} + 1)}{2} + \frac{(\mathbf{NVERTEX} - \mathbf{MIN\_NCC}) \cdot (\mathbf{NVERTEX} - \mathbf{MIN\_NCC} + 1)}{2} \tag{3.113}$$

$$\mathbf{arc\_gen} = \mathit{CLIQUE}(\geq) : \mathbf{NARC} \leq \frac{\mathbf{MIN\_NCC} \cdot (\mathbf{MIN\_NCC} + 1)}{2} + \frac{(\mathbf{NVERTEX} - \mathbf{MIN\_NCC}) \cdot (\mathbf{NVERTEX} - \mathbf{MIN\_NCC} + 1)}{2} \tag{3.114}$$

$$\mathbf{arc\_gen} = \mathit{CLIQUE}(<) : \mathbf{NARC} \leq \frac{\mathbf{MIN\_NCC} \cdot (\mathbf{MIN\_NCC} - 1)}{2} + \frac{(\mathbf{NVERTEX} - \mathbf{MIN\_NCC}) \cdot (\mathbf{NVERTEX} - \mathbf{MIN\_NCC} - 1)}{2} \tag{3.115}$$

$$\mathbf{arc\_gen} = \mathit{CLIQUE}(>) : \mathbf{NARC} \leq \frac{\mathbf{MIN\_NCC} \cdot (\mathbf{MIN\_NCC} - 1)}{2} + \frac{(\mathbf{NVERTEX} - \mathbf{MIN\_NCC}) \cdot (\mathbf{NVERTEX} - \mathbf{MIN\_NCC} - 1)}{2} \tag{3.116}$$

$$\mathbf{arc\_gen} = \mathit{CLIQUE}(\neq) : \mathbf{NARC} \leq \mathbf{MIN\_NCC}^2 - \mathbf{MIN\_NCC} + (\mathbf{NVERTEX} - \mathbf{MIN\_NCC})^2 - (\mathbf{NVERTEX} - \mathbf{MIN\_NCC}) \tag{3.117}$$

$$\mathbf{arc\_gen} = CYCLE : \mathbf{NARC} \leq \mathbf{NVERTEX} - 4 \cdot (\mathbf{MIN\_NCC} < \mathbf{NVERTEX}) \tag{3.118}$$

$$\mathbf{arc\_gen} = PATH : \mathbf{NARC} \leq \max(0, \mathbf{MIN\_NCC} - 1) + \max(0, \mathbf{NVERTEX} - \mathbf{MIN\_NCC} - 1) \tag{3.119}$$

Proof. (3.110) The maximum number of vertices according to a fixed number of vertices $\mathbf{NVERTEX}$ and to the fact that there is a connected component with $\mathbf{MIN\_NCC}$ vertices is obtained by:

- Building a connected component with $\mathbf{MIN\_NCC}$ vertices and creating an arc between each pair of vertices.
- Building a connected component with all the $\mathbf{NVERTEX} - \mathbf{MIN\_NCC}$ remaining vertices and creating an arc between each pair of vertices.

□

Proposition 84.

$$\mathbf{MIN\_NCC} > 1 \Rightarrow$$
$$\mathbf{NARC} \geq \left\lfloor \frac{\mathbf{NVERTEX}}{\mathbf{MIN\_NCC}} \right\rfloor \cdot (\mathbf{MIN\_NCC} - 1) + \mathbf{NVERTEX} \bmod \mathbf{MIN\_NCC} \tag{3.120}$$

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 $\mathbf{MIN\_NCC}$ vertices we get $\lfloor \frac{\mathbf{NVERTEX}}{\mathbf{MIN\_NCC}} \rfloor$ connected components.
- Since we can't build a connected component with the rest of the vertices (i.e. $\mathbf{NVERTEX} \bmod \mathbf{MIN\_NCC}$ vertices left) we have to incorporate them in the previous connected components and this costs one arc for each vertex.

□

When $\mathbf{MIN\_NCC} = 1$, note that Proposition 50 provides a lower bound on the number of arcs.

MIN_NCC, NCC, NVERTEX

Proposition 85.

$$\mathbf{NVERTEX} \geq \mathbf{NCC} \cdot \mathbf{MIN\_NCC} \tag{3.121}$$

Proof. The smallest number of vertices is obtained by taking all connected components to their minimum number of vertices $\mathbf{MIN\_NCC}$. □

MIN_NSCC, NARC, NVERTEX

Proposition 86.

$$\mathbf{NARC} \leq \mathbf{NVERTEX}^2 + \mathbf{MIN\_NSCC}^2 - \mathbf{NVERTEX} \cdot \mathbf{MIN\_NSCC} \quad (3.122)$$

Proof. Achieving the maximum number of arcs, provided that we have at least one strongly connected component with $\mathbf{MIN\_NSCC}$ vertices, is done by:

- Building a first strongly connected component $\mathcal{C}_1$ with $\mathbf{MIN\_NSCC}$ vertices and adding an arc between each pair of vertices of $\mathcal{C}_1$.
- Building a second strongly connected component $\mathcal{C}_2$ with $\mathbf{NVERTEX} - \mathbf{MIN\_NSCC}$ 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 $\mathbf{MIN\_NSCC}^2 + (\mathbf{NVERTEX} - \mathbf{MIN\_NSCC})^2 + \mathbf{MIN\_NSCC} \cdot (\mathbf{NVERTEX} - \mathbf{MIN\_NSCC})$. □

MIN_NSCC, NSCC, NVERTEX

Proposition 87.

$$\mathbf{NVERTEX} \geq \mathbf{NSCC} \cdot \mathbf{MIN\_NSCC} \quad (3.123)$$

Proof. Since each strongly connected component contains at least $\mathbf{MIN\_NSCC}$ vertices the total number of vertices is greater than or equal to $\mathbf{NSCC} \cdot \mathbf{MIN\_NSCC}$. □

NARC, NCC, NVERTEX

Proposition 88.

$$\mathbf{NARC} \leq (\mathbf{NVERTEX} - \mathbf{NCC} + 1)^2 + \mathbf{NCC} - 1 \quad (3.124)$$

$$\mathbf{arc\_gen} = \mathit{CIRCUIT} : \mathbf{NARC} \leq \mathbf{NVERTEX} - \mathbf{NCC} + 1 - (\mathbf{NCC} \neq 1) \quad (3.125)$$

$$\mathbf{arc\_gen} = \mathit{CHAIN} : \mathbf{NARC} \leq 2 \cdot \mathbf{NVERTEX} - 2 \cdot \mathbf{NCC} \quad (3.126)$$

$$\mathbf{arc\_gen} = \mathit{CLIQUE}(\leq) : \mathbf{NARC} \leq \mathbf{NCC} - 1 + \frac{(\mathbf{NVERTEX} - \mathbf{NCC} + 1) \cdot (\mathbf{NVERTEX} - \mathbf{NCC} + 2)}{2} \quad (3.127)$$

$$\mathbf{arc\_gen} = \mathit{CLIQUE}(\geq) : \mathbf{NARC} \leq \mathbf{NCC} - 1 + \frac{(\mathbf{NVERTEX} - \mathbf{NCC} + 1) \cdot (\mathbf{NVERTEX} - \mathbf{NCC} + 2)}{2} \quad (3.128)$$

$$\mathbf{arc\_gen} = \mathit{CLIQUE}(<) : \mathbf{NARC} \leq \mathbf{NCC} - 1 + \frac{(\mathbf{NVERTEX} - \mathbf{NCC} + 1) \cdot (\mathbf{NVERTEX} - \mathbf{NCC})}{2} \quad (3.129)$$

$$\mathbf{arc\_gen} = \mathit{CLIQUE}(>) : \mathbf{NARC} \leq \mathbf{NCC} - 1 + \frac{(\mathbf{NVERTEX} - \mathbf{NCC} + 1) \cdot (\mathbf{NVERTEX} - \mathbf{NCC})}{2} \quad (3.130)$$

$$\mathbf{arc\_gen} = CLIQUE(\neq) : \mathbf{NARC} \leq \max(0, \mathbf{NCC} - 1) + (\mathbf{NVERTEX} - \mathbf{NCC} + 1)^2 - (\mathbf{NVERTEX} - \mathbf{NCC} + 1) \tag{3.131}$$

$$\mathbf{arc\_gen} = CYCLE : \mathbf{NARC} \leq 2 \cdot \mathbf{NVERTEX} - 2 \cdot \mathbf{NCC} + 2 \cdot (\mathbf{NCC} = 1) \tag{3.132}$$

$$\mathbf{arc\_gen} = PATH : \mathbf{NARC} = \mathbf{NVERTEX} - \mathbf{NCC} \tag{3.133}$$

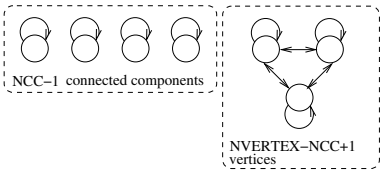


Figure 3.2: Illustration of Proposition 88. A graph that achieves the maximum number of arcs according to a fixed number of connected components as well as to a fixed number of vertices ($\mathbf{NCC} = 5, \mathbf{NVERTEX} = 7, \mathbf{NARC} = (7 - 5 + 1)^2 + 5 - 1 = 13$)

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. □

Proposition 89.

$$\mathbf{NARC} \geq \mathbf{NVERTEX} - \mathbf{NCC} \tag{3.134}$$

$$\texttt{equivalence} : \mathbf{NCC} > 0 \Rightarrow \mathbf{NARC} \geq (\mathbf{NVERTEX} \bmod \mathbf{NCC}) \cdot \left(\left\lfloor \frac{\mathbf{NVERTEX}}{\mathbf{NCC}} \right\rfloor + 1\right)^2 + (\mathbf{NCC} - \mathbf{NVERTEX} \bmod \mathbf{NCC}) \cdot \left\lfloor \frac{\mathbf{NVERTEX}}{\mathbf{NCC}} \right\rfloor^2 \tag{3.135}$$

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. □

NARC, NSCC, NVERTEX

Proposition 90.

$$\mathbf{NARC} \leq (\mathbf{NVERTEX}-\mathbf{NSCC}+1)\cdot\mathbf{NVERTEX}+\frac{\mathbf{NSCC}\cdot(\mathbf{NSCC}-1)}{2} \quad (3.136)$$

$$\texttt{equivalence} : \mathbf{NARC} \leq \mathbf{NSCC} - 1 + (\mathbf{NVERTEX} - \mathbf{NSCC} + 1)^2 \quad (3.137)$$

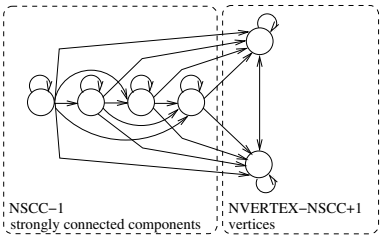


Figure 3.3: Illustration of Proposition 90(3.136). A graph that achieves the maximum number of arcs according to a fixed number of strongly connected components as well as to a fixed number of vertices ($\mathbf{NSCC} = 5, \mathbf{NVERTEX} = 6, \mathbf{NARC} = (6-5+1)\cdot 6+\frac{5\cdot(5-1)}{2} = 22$)

Proof. For proving 3.136, it is easier to rewrite the formula as $\mathbf{NARC} \leq (\mathbf{NVERTEX} - (\mathbf{NSCC}-1))^2 + (\mathbf{NCC}-1)\cdot(\mathbf{NVERTEX} - (\mathbf{NSCC}-1)) + \frac{\mathbf{NSCC}\cdot(\mathbf{NSCC}-1)}{2}$. 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 stricly 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. □

Proposition 91.

$$\mathbf{NARC} \geq \mathbf{NVERTEX} - \left\lfloor \frac{\mathbf{NSCC} - 1}{2} \right\rfloor \tag{3.138}$$

$$\begin{aligned} &\texttt{equivalence} : \mathbf{NSCC} > 0 \Rightarrow \\ &\mathbf{NARC} \geq (\mathbf{NVERTEX} \bmod \mathbf{NSCC}) \cdot \left(\left\lfloor \tfrac{\mathbf{NVERTEX}}{\mathbf{NSCC}} \right\rfloor + 1\right)^2 + \\ &(\mathbf{NSCC} - \mathbf{NVERTEX} \bmod \mathbf{NSCC}) \cdot \left\lfloor \tfrac{\mathbf{NVERTEX}}{\mathbf{NSCC}} \right\rfloor^2 \end{aligned} \tag{3.139}$$

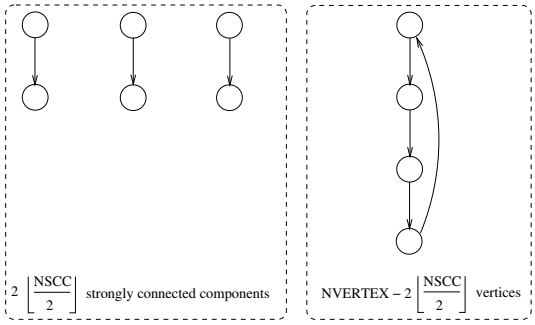


Figure 3.4: Illustration of Proposition 3.138. A graph that achieves the minimum number of arcs according to a fixed number of strongly connected components as well as to a fixed number of vertices ($\mathbf{NSCC} = 7, \mathbf{NVERTEX} = 10, \mathbf{NARC} = 10 - \lfloor \frac{7}{2} \rfloor = 7$)

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. □

Proposition 92.

$$\texttt{equivalence} : \mathbf{NVERTEX} > 0 \Rightarrow \mathbf{NSCC} \geq \left\lceil \frac{\mathbf{NVERTEX}^2}{\mathbf{NARC}} \right\rceil \tag{3.140}$$

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]. □

NARC, NSINK, NVERTEX

Proposition 93.

$$\mathbf{NARC} \leq (\mathbf{NVERTEX} - \mathbf{NSINK}) \cdot \mathbf{NVERTEX} \tag{3.141}$$

Proof. The maximum number of arcs is achieved by the following pattern: For all non-sink vertices we have an arc to all vertices. □

Proposition 94.

$$\mathbf{NARC} \geq \mathbf{NSINK} + max(0, \mathbf{NVERTEX} - 2 \cdot \mathbf{NSINK}) \tag{3.142}$$

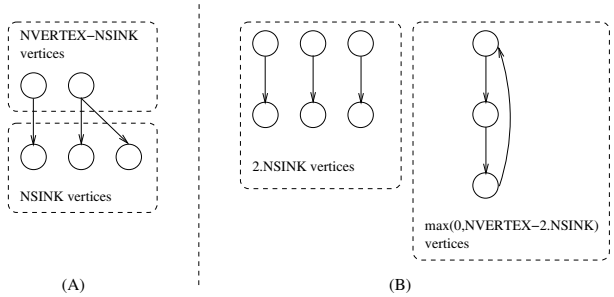


Figure 3.5: Illustration of Proposition 94. Graphs that achieve the minimum number of arcs according to a fixed number of sinks as well as to a fixed number of vertices (A : $\mathbf{NSINK} = 3, \mathbf{NVERTEX} = 5, \mathbf{NARC} = 3 + max(0, 5 - 2 \cdot 3) = 3; B$: $\mathbf{NSINK} = 3, \mathbf{NVERTEX} = 9, \mathbf{NARC} = 3 + max(0, 9 - 2 \cdot 3) = 6$)

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 $\mathbf{NARC}(G) \geq |V(G)| - \mathbf{NSINK}(G)$. □

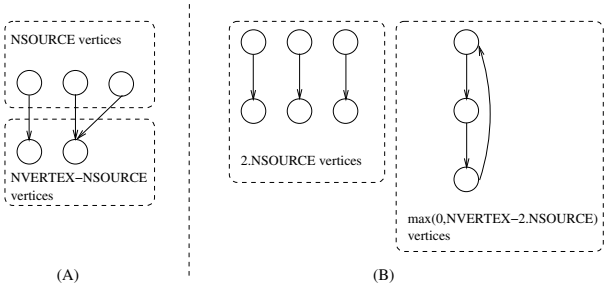


Figure 3.6: Illustration of Proposition 96. Graphs that achieve the minimum number of arcs according to a fixed number of sources as well as to a fixed number of vertices (A : $\mathbf{NSOURCE} = 3, \mathbf{NVERTEX} = 5, \mathbf{NARC} = 3 + max(0, 5 - 2 \cdot 3) = 3; B$: $\mathbf{NSOURCE} = 3, \mathbf{NVERTEX} = 9, \mathbf{NARC} = 3 + max(0, 9 - 2 \cdot 3) = 6$)

NARC, NSOURCE, NVERTEX

Proposition 95.

$$\mathbf{NARC} \leq (\mathbf{NVERTEX} - \mathbf{NSOURCE}) \cdot \mathbf{NVERTEX} \tag{3.143}$$

Proof. The maximum number of arcs is achieved by the following pattern: For all non-source vertices we have an arc from all vertices. □

Proposition 96.

$$\mathbf{NARC} \geq \mathbf{NSOURCE} + max(0, \mathbf{NVERTEX} - 2 \cdot \mathbf{NSOURCE}) \tag{3.144}$$

Proof. Similar to Proposition 94. □

NSINK, NSOURCE, NVERTEX

Proposition 97.

$$\mathbf{NVERTEX} \geq \mathbf{NSOURCE} + \mathbf{NSINK} \tag{3.145}$$

Proof. No vertex can be both a source and a sink (isolated vertices are removed). □

Graph invariants involving four characteristics of a final graph

$\boxed{\mathbf{MAX\_NCC, MIN\_NCC, NARC, NCC}}$

Proposition 98. *Let* α *denote* $\max(0, \mathbf{NCC} - 1)$.

$$\mathbf{NARC} \leq \alpha \cdot \mathbf{MAX\_NCC}^2 + \mathbf{MIN\_NCC}^2 \tag{3.146}$$

$$\mathbf{arc\_gen} = CIRCUIT : \mathbf{NARC} \leq \alpha \cdot \mathbf{MAX\_NCC} + \mathbf{MIN\_NCC} \tag{3.147}$$

$$\mathbf{arc\_gen} = CHAIN : \mathbf{NARC} \leq \alpha \cdot (2 \cdot \mathbf{MAX\_NCC} - 2) + 2 \cdot \mathbf{MIN\_NCC} - 2 \tag{3.148}$$

$$\mathbf{arc\_gen} \in \{CLIQUE(\leq), CLIQUE(\geq)\} : \mathbf{NARC} \leq \alpha \cdot \frac{\mathbf{MAX\_NCC} \cdot (\mathbf{MAX\_NCC}+1)}{2} + \frac{\mathbf{MIN\_NCC} \cdot (\mathbf{MIN\_NCC}+1)}{2} \tag{3.149}$$

$$\mathbf{arc\_gen} \in \{CLIQUE(<), CLIQUE(>)\} : \mathbf{NARC} \leq \alpha \cdot \frac{\mathbf{MAX\_NCC} \cdot (\mathbf{MAX\_NCC}-1)}{2} + \frac{\mathbf{MIN\_NCC} \cdot (\mathbf{MIN\_NCC}-1)}{2} \tag{3.150}$$

$$\mathbf{arc\_gen} = CLIQUE(\neq) : \mathbf{NARC} \leq \mathbf{MIN\_NCC}^2 - \mathbf{MIN\_NCC} + \alpha \cdot (\mathbf{MAX\_NCC}^2 - \mathbf{MAX\_NCC}) \tag{3.151}$$

$$\mathbf{arc\_gen} = CYCLE : \mathbf{NARC} \leq 2 \cdot \alpha \cdot \mathbf{MAX\_NCC} + 2 \cdot \mathbf{MIN\_NCC} \tag{3.152}$$

$$\mathbf{arc\_gen} = PATH : \mathbf{NARC} \leq \alpha \cdot (\mathbf{MAX\_NCC} - 1) + \mathbf{MIN\_NCC} - 1 \tag{3.153}$$

Proof. We construct $\mathbf{NCC} - 1$ connected components with $\mathbf{MAX\_NCC}$ vertices and one connected component with $\mathbf{MIN\_NCC}$ 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 $PATH$. □

Proposition 99.

$$\mathbf{NCC} > 0 \Rightarrow \mathbf{NARC} \geq (\mathbf{NCC} - 1) \cdot \max(1, \mathbf{MIN\_NCC} - 1) + \max(1, \mathbf{MAX\_NCC} - 1) \tag{3.154}$$

$$\mathbf{arc\_gen} = PATH : \mathbf{NARC} \geq \max(0, \mathbf{NCC} - 1) \cdot (\mathbf{MIN\_NCC} - 1) + \mathbf{MAX\_NCC} - 1 \tag{3.155}$$

Proof. (3.154) We construct $\mathbf{NCC} - 1$ connected components with $\mathbf{MIN\_NCC}$ vertices and one connected component with $\mathbf{MAX\_NCC}$ vertices. The quantity $\max(1, n-1)$ corresponds to the minimum number of arcs in a connected component of n $(n > 0)$ vertices. □

MAX_NCC, MIN_NCC, NCC, NVERTEX

Proposition 100.

$$\mathbf{NVERTEX} \leq \max(0, \mathbf{NCC} - 1) \cdot \mathbf{MAX\_NCC} + \mathbf{MIN\_NCC} \tag{3.156}$$

Proof. Derived from the definitions of $\mathbf{MIN\_NCC}$ and $\mathbf{MAX\_NCC}$. □

Proposition 101.

$$\mathbf{NVERTEX} \geq \max(0, \mathbf{NCC} - 1) \cdot \mathbf{MIN\_NCC} + \mathbf{MAX\_NCC} \tag{3.157}$$

Proof. Derived from the definitions of $\mathbf{MIN\_NCC}$ and $\mathbf{MAX\_NCC}$. □

MAX_NSCC, MIN_NSCC, NARC, NSCC

Proposition 102.

$$\begin{aligned}\mathbf{NARC} \leq\ & \max(0, \mathbf{NSCC} - 1) \cdot \mathbf{MAX\_NSCC}^2 + \mathbf{MIN\_NSCC}^2 + \\ & \max(0, \mathbf{NSCC} - 1) \cdot \mathbf{MIN\_NSCC} \cdot \mathbf{MAX\_NSCC} + \\ & \mathbf{MAX\_NSCC}^2 \cdot \tfrac{\max(0, \mathbf{NSCC}-2) \cdot \max(0, \mathbf{NSCC}-1)}{2}\end{aligned} \tag{3.158}$$

Proof. We assume that we have at least two strongly connected components (the case with one being obvious). Let $(SCC_i)_{i \in [\mathbf{NCC}(G)]}$ be the family of strongly connected components of G. Then $|E(G)| \leq \sum_{i \in [\mathbf{NCC}(G)]} |E(G[SCC_i])| + k$, 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, $k \leq \frac{1}{2} \cdot \sum_{i \in [\mathbf{NCC}(G)]} |SCC_i| \cdot (|V(G) - SCC_i|)$. W.l.o.g. we assume $|SCC_1| = \mathbf{MIN\_NCC}$. Then we get $k \leq \frac{1}{2} \cdot (\mathbf{MIN\_NCC} \cdot (\mathbf{NCC} - 1) \cdot \mathbf{MAX\_NCC} + \mathbf{MAX\_NCC} \cdot ((\mathbf{NCC} - 2) \cdot \mathbf{MAX\_NCC} + \mathbf{MIN\_NCC}))$. □

Proposition 103.

$$\mathbf{NARC} \geq \max(0, \mathbf{NSCC} - 1) \cdot \mathbf{MIN\_NSCC} + \mathbf{MAX\_NSCC} \tag{3.159}$$

Proof. Let $(SCC_i)_{i \in [\mathbf{NCC}(G)]}$ be the family of strongly connected components of G, as $|E(G)| \geq \sum_{i \in [\mathbf{NCC}(G)]} |E(G[SCC_i])|$, we obtain the result since in a strongly connected graph the number of edges is at least its number of vertices. □

MAX_NSCC, MIN_NSCC, NSCC, NVERTEX

Proposition 104.

$$\mathbf{NVERTEX} \leq \max(0, \mathbf{NSCC} - 1) \cdot \mathbf{MAX\_NSCC} + \mathbf{MIN\_NSCC} \tag{3.160}$$

Proof. Derived from the definitions of $\mathbf{MIN\_NSCC}$ and $\mathbf{MAX\_NSCC}$. □

Proposition 105.

$$\mathbf{NVERTEX} \geq \max(0, \mathbf{NSCC} - 1) \cdot \mathbf{MIN\_NSCC} + \mathbf{MAX\_NSCC} \tag{3.161}$$

Proof. Derived from the definitions of $\mathbf{MIN\_NSCC}$ and $\mathbf{MAX\_NSCC}$. □

MIN_NCC, NARC, NCC, NVERTEX

Proposition 106. *Let α, β and γ respectively denote* $\max(0, \mathbf{NCC} - 1)$, $\mathbf{NVERTEX} - \alpha \cdot \mathbf{MIN\_NCC}$ *and* $\mathbf{MIN\_NCC}$.

$$\mathbf{NARC} \leq \alpha \cdot \gamma^2 + \beta^2 \tag{3.162}$$

$$\mathbf{arc\_gen} \in \{CLIQUE(\leq), CLIQUE(\geq)\} : \mathbf{NARC} \leq \alpha \cdot \frac{\gamma \cdot (\gamma + 1)}{2} + \frac{\beta \cdot (\beta + 1)}{2} \tag{3.163}$$

$$\mathbf{arc\_gen} \in \{CLIQUE(<), CLIQUE(>)\} : \mathbf{NARC} \leq \alpha \cdot \frac{\gamma \cdot (\gamma - 1)}{2} + \frac{\beta \cdot (\beta - 1)}{2} \tag{3.164}$$

$$\mathbf{arc\_gen} = CLIQUE(\neq) : \mathbf{NARC} \leq \alpha \cdot \gamma \cdot (\gamma - 1) + \beta \cdot (\beta - 1) \tag{3.165}$$

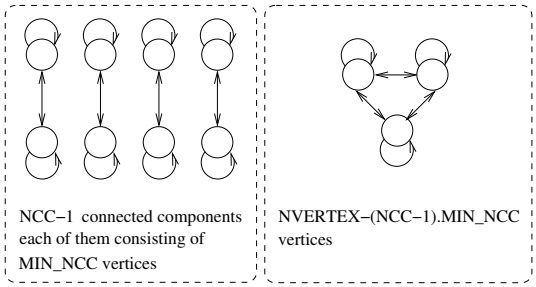


Figure 3.7: Illustration of Proposition 106(3.162). Graphs that achieve the maximum number of arcs according to a minimum number of vertices in a connected component, to a number of connected components, as well as to a fixed number of vertices ($\mathbf{MIN\_NCC} = 2, \mathbf{NCC} = 5, \mathbf{NVERTEX} = 11, \mathbf{NARC} = (11 - (5-1) \cdot 2)^2 + (5-1) \cdot 2^2 = 25$)

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. □

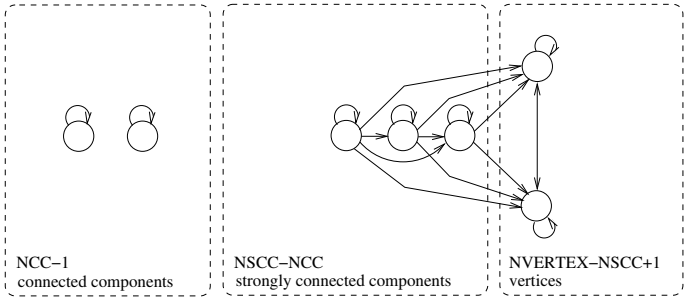


Figure 3.8: Illustration of Proposition 107. A graph that achieves the maximum number of arcs according to a fixed number of connected components, to a fixed number of strongly connected components as well as to a fixed number of vertices ($\mathbf{NCC} = 3, \mathbf{NSCC} = 6, \mathbf{NVERTEX} = 7, \mathbf{NARC} = 3 - 1 + (7 - 6 + 1)(7 - 3 + 1) + (6 - 3 + 1)(6 - 3)/2 = 18$)

NARC, NCC, NSCC, NVERTEX

Proposition 107.

$$\mathbf{NARC} \leq \mathbf{NCC} - 1 + (\mathbf{NVERTEX} - \mathbf{NSCC} + 1) \cdot (\mathbf{NVERTEX} - \mathbf{NCC} + 1) + \frac{(\mathbf{NSCC} - \mathbf{NCC} + 1) \cdot (\mathbf{NSCC} - \mathbf{NCC})}{2} \tag{3.166}$$

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.

$$\mathbf{NARC} \geq \mathbf{NVERTEX} - \max(0, \min(\mathbf{NCC}, \mathbf{NSCC} - \mathbf{NCC})) \tag{3.167}$$

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.

$$\begin{aligned}\mathbf{NARC} \leq \mathbf{NVERTEX}^2 &- \mathbf{NVERTEX} \cdot \mathbf{NSOURCE} \\ &- \mathbf{NVERTEX} \cdot \mathbf{NSINK} + \mathbf{NSOURCE} \cdot \mathbf{NSINK}\end{aligned} \tag{3.168}$$

Proof. Since the maximum number of arcs of a digraph is $\mathbf{NVERTEX}^2$, and since:

- No vertex can have a source as a successor we lose $\mathbf{NVERTEX} \cdot \mathbf{NSOURCE}$ arcs,
- No sink can have a successor we lose $\mathbf{NVERTEX} \cdot \mathbf{NSINK}$ 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. □

Graph invariants involving five characteristics of a final graph

MAX_NCC, MIN_NCC, NARC, NCC, NVERTEX

Proposition 110.
Let:

- $\Delta = \mathbf{NVERTEX} - \mathbf{NCC} \cdot \mathbf{MIN\_NCC}$,
- $\delta = \lfloor \frac{\Delta}{\max(1, \mathbf{MAX\_NCC} - \mathbf{MIN\_NCC})} \rfloor$,
- $r = \Delta \bmod \max(1, \mathbf{MAX\_NCC} - \mathbf{MIN\_NCC})$,
- $\epsilon = (r > 0)$.

$$\Delta = 0 \vee (\mathbf{MAX\_NCC} \neq \mathbf{MIN\_NCC} \wedge \delta + \epsilon \leq \mathbf{NCC}) \tag{3.169}$$

$$\mathbf{NARC} \leq (\mathbf{NCC} - \delta - \epsilon) \cdot \mathbf{MIN\_NCC}^2 + \epsilon \cdot (\mathbf{MIN\_NCC} + r)^2 + \delta \cdot \mathbf{MAX\_NCC}^2 \tag{3.170}$$

Proposition 110 is currently a conjecture.

MIN_NCC, NARC, NCC, NSCC, NVERTEX

Proposition 111.

$$\begin{aligned}\mathbf{NARC} \leq &(\mathbf{NCC} - 1) \cdot \max(1, (\mathbf{MIN\_NCC} - 1)) + \\ &(\mathbf{NVERTEX} - \mathbf{NSCC} + 1) \cdot (\mathbf{NVERTEX} - \mathbf{NCC} + 1) + \\ &\frac{(\mathbf{NSCC} - \mathbf{NCC} + 1) \cdot (\mathbf{NSCC} - \mathbf{NCC})}{2}\end{aligned} \tag{3.171}$$

Proposition 111 is currently a conjecture.

Graph invariants relating two characteristics of two final graphs

$\boxed{\mathbf{MAX\_NCC}_1, \mathbf{NCC}_2}$

Proposition 112.

$$\texttt{vpartition} : \mathbf{MAX\_NCC}_1 < \mathbf{NVERTEX}_{\texttt{INITIAL}} \Leftrightarrow \mathbf{NCC}_2 > 0 \tag{3.172}$$

$$\texttt{apartition} : \mathbf{MAX\_NCC}_1 < \mathbf{NVERTEX}_{\texttt{INITIAL}} \Leftrightarrow \mathbf{NCC}_2 > 0 \tag{3.173}$$

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. □

$\boxed{\mathbf{MAX\_NCC}_2, \mathbf{NCC}_1}$

Proposition 113.

$$\texttt{vpartition} : \mathbf{MAX\_NCC}_2 < \mathbf{NVERTEX}_{\texttt{INITIAL}} \Leftrightarrow \mathbf{NCC}_1 > 0 \tag{3.174}$$

$$\texttt{apartition} : \mathbf{MAX\_NCC}_2 < \mathbf{NVERTEX}_{\texttt{INITIAL}} \Leftrightarrow \mathbf{NCC}_1 > 0 \tag{3.175}$$

Proof. Similar to Proposition 112. □

$\boxed{\mathbf{MIN\_NCC}_1, \mathbf{NCC}_2}$

Proposition 114.

$$\texttt{vpartition} : \mathbf{MIN\_NCC}_1 < \mathbf{NVERTEX}_{\texttt{INITIAL}} \Leftrightarrow \mathbf{NCC}_2 > 0 \tag{3.176}$$

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.

□

$\boxed{\mathbf{MIN\_NCC}_2, \mathbf{NCC}_1}$

Proposition 115.

$$\texttt{vpartition} : \mathbf{MIN\_NCC}_2 < \mathbf{NVERTEX}_{\texttt{INITIAL}} \Leftrightarrow \mathbf{NCC}_1 > 0 \tag{3.177}$$

Proof. Similar to Proposition 114. □

$\boxed{\mathbf{NARC}_1, \mathbf{NARC}_2}$

Proposition 116.

$$\texttt{apartition} \wedge \mathbf{arc\_gen} = PATH : \mathbf{NARC}_1 + \mathbf{NARC}_2 = \mathbf{NVERTEX}_{\texttt{INITIAL}} - 1 \tag{3.178}$$

Proof. Holds since each arc of the initial graph belongs to one of the two final graphs and since the initial graph has $\mathbf{NVERTEX}_{\texttt{INITIAL}} - 1$ arcs. □

$\boxed{\mathbf{NCC}_1, \mathbf{NCC}_2}$

Proposition 117.

$$\texttt{apartition} \wedge \mathbf{arc\_gen} = PATH : |\mathbf{NCC}_1 - \mathbf{NCC}_2| \leq 1 \tag{3.179}$$

$$\texttt{vpartition} \wedge \texttt{consecutive\_loops\_are\_connected} : |\mathbf{NCC}_1 - \mathbf{NCC}_2| \leq 1 \tag{3.180}$$

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.

$$\texttt{apartition} \wedge \mathbf{arc\_gen} = PATH : \mathbf{NCC}_1 + \mathbf{NCC}_2 < \mathbf{NVERTEX}_{\texttt{INITIAL}} \tag{3.181}$$

Proof. Holds because the initial graph is a path. □

$\boxed{\mathbf{NVERTEX}_1, \mathbf{NVERTEX}_2}$

Proposition 119.

$$\texttt{vpartition} : \mathbf{NVERTEX}_1 + \mathbf{NVERTEX}_2 = \mathbf{NVERTEX}_{\texttt{INITIAL}} \tag{3.182}$$

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

$\boxed{\mathbf{MAX\_NCC}_1, \mathbf{MIN\_NCC}_1, \mathbf{MIN\_NCC}_2}$

Proposition 120.

$$\begin{aligned}&\texttt{apartition} \wedge \mathbf{arc\_gen} = PATH :\\ &\max(2, \mathbf{MIN\_NCC}_1) + \max(3, \mathbf{MIN\_NCC}_1 + 1, \mathbf{MAX\_NCC}_1) +\\ &\max(2, \mathbf{MIN\_NCC}_2) - 2 > \mathbf{NVERTEX}_{\texttt{INITIAL}} \Rightarrow \mathbf{MIN\_NCC}_1 = \mathbf{MAX\_NCC}_1\end{aligned} \tag{3.183}$$

Proof. The quantity $\max(2, \mathbf{MIN\_NCC}_1) + \max(3, \mathbf{MIN\_NCC}_1 + 1, \mathbf{MAX\_NCC}_1) + \max(2, \mathbf{MIN\_NCC}_2) - 2$ corresponds to the minimum number of variables needed for building two non-empty connected components of respective size $\mathbf{MIN\_NCC}_1$ and $\mathbf{MAX\_NCC}_1$ such that $\mathbf{MAX\_NCC}_1$ is strictly greater than $\mathbf{MIN\_NCC}_1$. If this quantity is greater than the total number of variables we have that $\mathbf{MIN\_NCC}_1 = \mathbf{MAX\_NCC}_1$. □

Proposition 121.

$$\begin{aligned}&\texttt{vpartition} \wedge \texttt{consecutive\_loops\_are\_connected} :\\ &\max(1, \mathbf{MIN\_NCC}_1) + \max(2, \mathbf{MIN\_NCC}_1 + 1, \mathbf{MAX\_NCC}_1) +\\ &\max(1, \mathbf{MIN\_NCC}_2) > \mathbf{NVERTEX}_{\texttt{INITIAL}} \Rightarrow \mathbf{MIN\_NCC}_1 = \mathbf{MAX\_NCC}_1\end{aligned} \tag{3.184}$$

Proof. The quantity $\max(1, \mathbf{MIN\_NCC}_1) + \max(2, \mathbf{MIN\_NCC}_1 + 1, \mathbf{MAX\_NCC}_1) + \max(1, \mathbf{MIN\_NCC}_2)$ corresponds to the minimum number of variables needed for building two non-empty connected components of respective size $\mathbf{MIN\_NCC}_1$ and $\mathbf{MAX\_NCC}_1$ such that $\mathbf{MAX\_NCC}_1$ is strictly greater than $\mathbf{MIN\_NCC}_1$. If this quantity is greater than the total number of variables we have that $\mathbf{MIN\_NCC}_1 = \mathbf{MAX\_NCC}_1$. □

$\boxed{\mathbf{MAX\_NCC}_2, \mathbf{MIN\_NCC}_2, \mathbf{MIN\_NCC}_1}$

Proposition 122.

$$\begin{aligned}&\texttt{apartition} \wedge \mathbf{arc\_gen} = PATH :\\ &\max(2, \mathbf{MIN\_NCC}_2) + \max(3, \mathbf{MIN\_NCC}_2 + 1, \mathbf{MAX\_NCC}_2) +\\ &\max(2, \mathbf{MIN\_NCC}_1) - 2 > \mathbf{NVERTEX}_{\texttt{INITIAL}} \Rightarrow \mathbf{MIN\_NCC}_2 = \mathbf{MAX\_NCC}_2\end{aligned} \tag{3.185}$$

Proof. Similar to Proposition 120. □

Proposition 123.

$$\begin{aligned}&\texttt{vpartition} \wedge \texttt{consecutive\_loops\_are\_connected} :\\ &\max(1, \mathbf{MIN\_NCC}_2) + \max(2, \mathbf{MIN\_NCC}_2 + 1, \mathbf{MAX\_NCC}_2) +\\ &\max(1, \mathbf{MIN\_NCC}_1) > \mathbf{NVERTEX}_{\texttt{INITIAL}} \Rightarrow \mathbf{MIN\_NCC}_2 = \mathbf{MAX\_NCC}_2\end{aligned} \tag{3.186}$$

Proof. Similar to Proposition 121. □

$\boxed{\mathbf{MIN\_NCC}_1, \mathbf{NARC}_2, \mathbf{NCC}_1}$

Proposition 124.

$$\begin{array}{l}\texttt{apartition} \wedge \mathbf{arc\_gen} = PATH \wedge \mathbf{NVERTEX}_{\texttt{INITIAL}} > 0 : \\ \quad \mathbf{NCC}_1 = 1 \Leftrightarrow \mathbf{MIN\_NCC}_1 + \mathbf{NARC}_2 = \mathbf{NVERTEX}_{\texttt{INITIAL}}\end{array} \tag{3.187}$$

Proof. When $\mathbf{MIN\_NCC}_1 + \mathbf{NARC}_2 = \mathbf{NVERTEX}_{\texttt{INITIAL}}$ there is no more room for an extra connected component for the first final graph. □

$\boxed{\mathbf{MIN\_NCC}_1, \mathbf{NARC}_2, \mathbf{NCC}_1}$

Proposition 125.

$$\begin{array}{l}\texttt{apartition} \wedge \mathbf{arc\_gen} = PATH \wedge \mathbf{NVERTEX}_{\texttt{INITIAL}} > 0 : \\ \quad \mathbf{NCC}_2 = 1 \Leftrightarrow \mathbf{MIN\_NCC}_2 + \mathbf{NARC}_1 = \mathbf{NVERTEX}_{\texttt{INITIAL}}\end{array} \tag{3.188}$$

Proof. Similar to Proposition 124. □

Graph invariants relating four characteristics of two final graphs

$\boxed{\mathbf{MAX\_NCC}_1, \mathbf{MIN\_NCC}_1, \mathbf{MIN\_NCC}_2, \mathbf{NCC}_1}$

Proposition 126.

$$\begin{array}{l}\texttt{apartition} \wedge \mathbf{arc\_gen} = PATH : \\ \max(2, \mathbf{MIN\_NCC}_1) + \max(2, \mathbf{MAX\_NCC}_1) + \max(2, \mathbf{MIN\_NCC}_2) - 2 > \\ \quad \mathbf{NVERTEX}_{\texttt{INITIAL}} \Rightarrow \mathbf{NCC}_1 \leq 1\end{array} \tag{3.189}$$

Proof. The quantity $\max(2, \mathbf{MIN\_NCC}_1) + \max(2, \mathbf{MAX\_NCC}_1) + \max(2, \mathbf{MIN\_NCC}_2) - 2$ corresponds to the minimum number of variables needed for building two non-empty connected components of respective size $\mathbf{MIN\_NCC}_1$ and $\mathbf{MAX\_NCC}_1$. If this quantity is greater than the total number of variables we have that $\mathbf{NCC}_1 \leq 1$. □

Proposition 127.

$$\begin{array}{l}\texttt{vpartition} \wedge \texttt{consecutive\_loops\_are\_connected} : \\ \max(1, \mathbf{MIN\_NCC}_1) + \max(1, \mathbf{MAX\_NCC}_1) + \max(1, \mathbf{MIN\_NCC}_2) > \\ \quad \mathbf{NVERTEX}_{\texttt{INITIAL}} \Rightarrow \mathbf{NCC}_1 \leq 1\end{array} \tag{3.190}$$

Proof. The quantity $\max(1, \mathbf{MIN\_NCC}_1) + \max(1, \mathbf{MAX\_NCC}_1) + \max(1, \mathbf{MIN\_NCC}_2)$ corresponds to the minimum number of variables needed for building two non-empty connected components of respective size $\mathbf{MIN\_NCC}_1$ and $\mathbf{MAX\_NCC}_1$. If this quantity is greater than the total number of variables we have that $\mathbf{NCC}_1 \leq 1$. □

$\boxed{\mathbf{MAX\_NCC}_2, \mathbf{MIN\_NCC}_2, \mathbf{MIN\_NCC}_1, \mathbf{NCC}_2}$

Proposition 128.

$$\begin{array}{l}\texttt{apartition} \wedge \mathbf{arc\_gen} = PATH : \\ \max(2, \mathbf{MIN\_NCC}_2) + \max(2, \mathbf{MAX\_NCC}_2) + \max(2, \mathbf{MIN\_NCC}_1) - 2 > \\ \quad \mathbf{NVERTEX}_{\texttt{INITIAL}} \Rightarrow \mathbf{NCC}_2 \leq 1\end{array} \tag{3.191}$$

Proof. Similar to Proposition 126. □

Proposition 129.

$$\begin{array}{l}\texttt{vpartition} \wedge \texttt{consecutive\_loops\_are\_connected} : \\ \max(1, \mathbf{MIN\_NCC}_2) + \max(1, \mathbf{MAX\_NCC}_2) + \max(1, \mathbf{MIN\_NCC}_1) > \\ \quad \mathbf{NVERTEX}_{\texttt{INITIAL}} \Rightarrow \mathbf{NCC}_2 \leq 1\end{array} \tag{3.192}$$

Proof. Similar to Proposition 127. □

Graph invariants relating five characteristics of two final graphs

$\boxed{\mathbf{MAX\_NCC}_1, \mathbf{MAX\_NCC}_2, \mathbf{MIN\_NCC}_1, \mathbf{MIN\_NCC}_2, \mathbf{NCC}_1}$

Proposition 130.

$$\begin{array}{l}\texttt{vpartition} \wedge \texttt{consecutive\_loops\_are\_connected} : \\ \mathbf{MIN\_NCC}_1 \cdot \max(0, \mathbf{NCC}_1 - 1) + \mathbf{MAX\_NCC}_1 + \\ \mathbf{MIN\_NCC}_2 \cdot \max(0, \mathbf{NCC}_1 - 2) + \mathbf{MAX\_NCC}_2 \leq \mathbf{NVERTEX}_{\texttt{INITIAL}}\end{array} \tag{3.193}$$

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. □

Proposition 131.

$$\begin{array}{l}\texttt{vpartition} \wedge \texttt{consecutive\_loops\_are\_connected} : \\ \overline{\mathbf{NCC}_1} \leq (\overline{\mathbf{MAX\_NCC}_1} > 0) + \left\lfloor \dfrac{\alpha}{\beta} \right\rfloor + \left(\alpha \bmod \beta \geq \max(1, \underline{\mathbf{MIN\_NCC}_1})\right) \\ \left\{ \begin{array}{l} \bullet\ \alpha = \max(0, \mathbf{NVERTEX}_{\texttt{INITIAL}} - \max(1, \underline{\mathbf{MAX\_NCC}_1}) - \max(1, \underline{\mathbf{MAX\_NCC}_2})), \\ \bullet\ \beta = \max(1, \underline{\mathbf{MIN\_NCC}_1}) + \max(1, \underline{\mathbf{MIN\_NCC}_2}). \end{array} \right.\end{array} \tag{3.194}$$

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 $\max(1, \underline{\mathbf{MAX\_NCC}_1})$ and $\max(1, \underline{\mathbf{MAX\_NCC}_2})$, which have to be built. □

Proposition 132.

$$\begin{array}{l}\texttt{vpartition} \wedge \texttt{consecutive\_loops\_are\_connected} : \\ \mathbf{MAX\_NCC}_1 \cdot \max(0, \mathbf{NCC}_1 - 1) + \mathbf{MIN\_NCC}_1 + \\ \mathbf{MAX\_NCC}_2 \cdot \mathbf{NCC}_1 + \mathbf{MIN\_NCC}_2 \geq \mathbf{NVERTEX}_{\texttt{INITIAL}}\end{array} \tag{3.195}$$

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. □

Proposition 133.

$$\texttt{vpartition} \wedge \texttt{consecutive\_loops\_are\_connected}:$$

$$\underline{\mathbf{NCC}_1} \geq (\overline{\mathbf{MAX\_NCC}_2} < \mathbf{NVERTEX}_{\texttt{INITIAL}}) + \left\lfloor \frac{\alpha}{\beta} \right\rfloor + \left(\alpha \bmod \beta > \overline{\mathbf{MAX\_NCC}_2}\right)$$

$$\begin{cases} \bullet\ \alpha = \max(0, \mathbf{NVERTEX}_{\texttt{INITIAL}} - \overline{\mathbf{MIN\_NCC}_1} - \overline{\mathbf{MIN\_NCC}_2}, \\ \bullet\ \beta = \max(1, \overline{\mathbf{MAX\_NCC}_1}) + \max(1, \overline{\mathbf{MAX\_NCC}_2}). \end{cases} \tag{3.196}$$

Proof. The minimum number of connected components is achieved by taking the groups as large as possible except for two groups of respective size $\overline{\mathbf{MIN\_NCC}_2}$ and $\overline{\mathbf{MIN\_NCC}_1}$, which have to be built. □

Proposition 134.

$$\begin{array}{l} \texttt{vpartition} \wedge \texttt{consecutive\_loops\_are\_connected}: \\ \mathbf{MAX\_NCC}_2 \leq \max(\mathbf{MIN\_NCC}_2, \mathbf{NVERTEX}_{\texttt{INITIAL}} - \alpha), \texttt{with}: \\ \bullet\ \alpha = \mathbf{MIN\_NCC}_1 \cdot \max(0, \mathbf{NCC}_1 - 1) + \mathbf{MAX\_NCC}_1 + \\ \qquad \mathbf{MIN\_NCC}_2 + \mathbf{MIN\_NCC}_2 \cdot \max(0, \mathbf{NCC}_1 - 3) \end{array} \tag{3.197}$$

Proof. If $\mathbf{NCC}_1 \leq 1$ we have that $\mathbf{MAX\_NCC}_2 \leq \mathbf{MIN\_NCC}_2$. Otherwise, when $\mathbf{NCC}_1 > 1$, we have that $\mathbf{MIN\_NCC}_1 \cdot \max(0, \mathbf{NCC}_1 - 1) + \mathbf{MAX\_NCC}_1 + \mathbf{MIN\_NCC}_2 + \mathbf{MAX\_NCC}_2 + \mathbf{MIN\_NCC}_2 \cdot \max(0, \mathbf{NCC}_1 - 3) \leq \mathbf{NVERTEX}_{\texttt{INITIAL}}$. $\mathbf{NCC}_1 - 3$ comes from the fact that we build the minimum number of connected components in the second final graph (i.e.$\mathbf{NCC}_1 - 1$ connected components) and that we have already built two connected components of respective size $\mathbf{MIN\_NCC}_2$ and $\mathbf{MAX\_NCC}_2$. By isolating $\mathbf{MAX\_NCC}_2$ in the previous expression and by grouping the two inequalities the result follows. □

Proposition 135.

$$\begin{array}{l} \texttt{apartition} \wedge \mathbf{arc\_gen} = \mathit{PATH} \wedge \mathbf{MIN\_NCC}_1 > 1 \wedge \mathbf{MIN\_NCC}_2 > 1: \\ \mathbf{NCC}_1 \leq (\mathbf{MAX\_NCC}_1 > 0) + \left\lfloor \frac{\alpha}{\beta} \right\rfloor + ((\alpha \bmod \beta) + 1 \geq \mathbf{MIN\_NCC}_1), \texttt{with}: \\ \begin{cases} \bullet\ \alpha = \max(0, \mathbf{NVERTEX}_{\texttt{INITIAL}} - \mathbf{MAX\_NCC}_1 - \mathbf{MAX\_NCC}_2 + 1), \\ \bullet\ \beta = \mathbf{MIN\_NCC}_1 + \mathbf{MIN\_NCC}_2 - 2. \end{cases} \end{array} \tag{3.198}$$

Proof. The maximum number of connected components of G_1 is achieved by:

- Building a first connected component of G_1 involving $\mathbf{MAX\_NCC}_1$ vertices,
- Building a first connected component of G_2 involving $\mathbf{MAX\_NCC}_2$ vertices,
- Building alternatively a connected component of G_1 and a connected component of G_2 involving respectively $\mathbf{MIN\_NCC}_1$ and $\mathbf{MIN\_NCC}_2$ vertices,
- Finally, if this is possible, building a connected component of G_1 involving $\mathbf{MIN\_NCC}_1$ vertices.

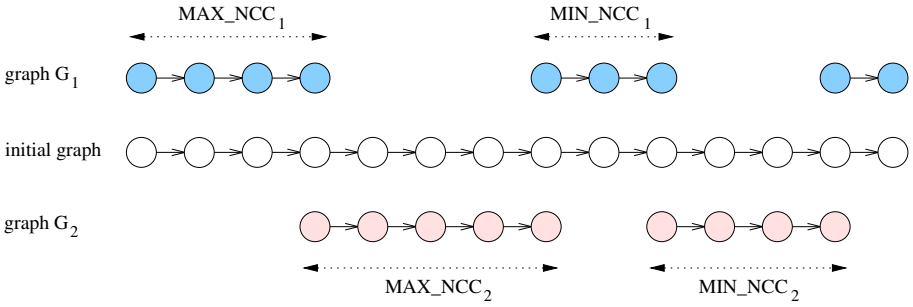


Figure 3.9: Illustration of Proposition 135. Configuration achieving the maximum number of connected components for G_1 according to the size of the smallest and largest connected components of G_1 and G_2 and to an initial number of vertices ($\mathbf{MAX\_NCC}_1 = 4, \mathbf{MAX\_NCC}_2 = 5, \mathbf{MIN\_NCC}_1 = 3, \mathbf{MIN\_NCC}_2 = 4, \mathbf{NVERTEX}_{\texttt{INITIAL}} = 14, \alpha = \max(0, 14 - 4 - 5 + 1) = 6, \beta = \max(2, 3 + 4 - 2) = 5, \mathbf{NCC}_1 = (4 > 0) + \lfloor \frac{6}{5} \rfloor + (((6 \bmod 5) + 1) \geq 3) = 2$)

□

Proposition 136.

$$\texttt{apartition} \wedge \mathbf{arc\_gen} = \mathit{PATH} \wedge \mathbf{MIN\_NCC}_1 > 1 \wedge \mathbf{MIN\_NCC}_2 > 1 :$$
$$\mathbf{NCC}_1 \geq (\mathbf{MIN\_NCC}_1 > 0) + \left\lfloor \frac{\alpha}{\beta} \right\rfloor + ((\alpha \bmod \beta) + 1 > \mathbf{MAX\_NCC}_2), \texttt{ with} :$$
$$\begin{cases} \bullet\ \alpha = \max(0, \mathbf{NVERTEX}_{\texttt{INITIAL}} - \mathbf{MIN\_NCC}_1 - \mathbf{MIN\_NCC}_2 + 1), \\ \bullet\ \beta = \mathbf{MAX\_NCC}_1 + \mathbf{MAX\_NCC}_2 - 2. \end{cases} \tag{3.199}$$

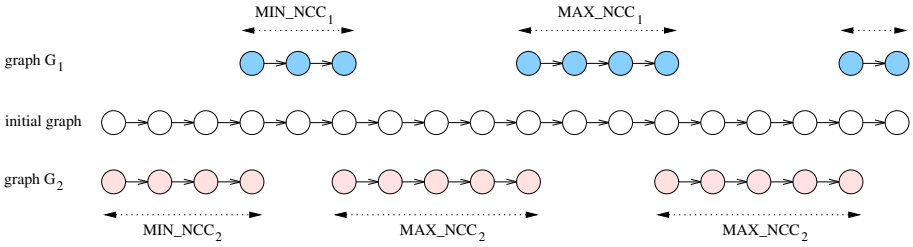


Figure 3.10: Illustration of Proposition 136. Configuration achieving the minimum number of connected components for G_1 according to the size of the smallest and largest connected components of G_1 and G_2 and to an initial number of vertices ($\mathbf{MAX\_NCC}_1 = 4, \mathbf{MAX\_NCC}_2 = 5, \mathbf{MIN\_NCC}_1 = 3, \mathbf{MIN\_NCC}_2 = 4, \mathbf{NVERTEX}_{\texttt{INITIAL}} = 18, \alpha = \max(0, 18 - 3 - 4 + 1) = 12, \beta = \max(2, 4 + 5 - 2) = 7, \mathbf{NCC}_1 = (3 > 0) + \lfloor \frac{12}{7} \rfloor + (((12 \bmod 7) + 1) > 5) = 3$)

Proof. The minimum number of connected components of G_1 is achieved by:

- Building a first connected component of G_2 involving $\mathbf{MIN\_NCC}_2$ vertices,
- Building a first connected component of G_1 involving $\mathbf{MIN\_NCC}_1$ vertices,

- Building alternatively a connected component of G_2 and a connected component of G_1 involving respectively $\mathbf{MAX\_NCC}_2$ and $\mathbf{MAX\_NCC}_1$ vertices,
- Finally, if this is possible, building a connected component of G_2 involving $\mathbf{MAX\_NCC}_2$ 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.

□

$\mathbf{MAX\_NCC}_1, \mathbf{MAX\_NCC}_2, \mathbf{MIN\_NCC}_1, \mathbf{MIN\_NCC}_2, \mathbf{NCC}_2$

Proposition 137.

$$\texttt{vpartition} \wedge \texttt{consecutive\_loops\_are\_connected}:$$
$$\mathbf{MIN\_NCC}_2 \cdot \max(0, \mathbf{NCC}_2 - 1) + \mathbf{MAX\_NCC}_2 + \quad (3.200)$$
$$\mathbf{MIN\_NCC}_1 \cdot \max(0, \mathbf{NCC}_2 - 2) + \mathbf{MAX\_NCC}_1 \leq \mathbf{NVERTEX}_{\texttt{INITIAL}}$$

Proof. Similar to Proposition 130. □

Proposition 138.

$$\texttt{vpartition} \wedge \texttt{consecutive\_loops\_are\_connected}:$$
$$\overline{\mathbf{NCC}_2} \leq (\overline{\mathbf{MAX\_NCC}_2} > 0) + \left\lfloor \frac{\alpha}{\beta} \right\rfloor + \left(\alpha \bmod \beta \geq \max(1, \underline{\mathbf{MIN\_NCC}_2})\right)$$
$$\begin{cases} \bullet\ \alpha = \max(0, \mathbf{NVERTEX}_{\texttt{INITIAL}} - \max(1, \underline{\mathbf{MAX\_NCC}_2}) - \max(1, \underline{\mathbf{MAX\_NCC}_1})), \\ \bullet\ \beta = \max(1, \underline{\mathbf{MIN\_NCC}_2}) + \max(1, \underline{\mathbf{MIN\_NCC}_1}). \end{cases} \quad (3.201)$$

Proof. Similar to Proposition 131. □

Proposition 139.

$$\texttt{vpartition} \wedge \texttt{consecutive\_loops\_are\_connected}:$$
$$\mathbf{MAX\_NCC}_2 \cdot \max(0, \mathbf{NCC}_2 - 1) + \mathbf{MIN\_NCC}_2 + \quad (3.202)$$
$$\mathbf{MAX\_NCC}_1 \cdot \mathbf{NCC}_2 + \mathbf{MIN\_NCC}_1 \geq \mathbf{NVERTEX}_{\texttt{INITIAL}}$$

Proof. Similar to Proposition 132. □

Proposition 140.

$$\texttt{vpartition} \wedge \texttt{consecutive\_loops\_are\_connected}:$$
$$\underline{\mathbf{NCC}_2} \geq (\overline{\mathbf{MAX\_NCC}_1} < \mathbf{NVERTEX}_{\texttt{INITIAL}}) + \left\lfloor \frac{\alpha}{\beta} \right\rfloor + \left(\alpha \bmod \beta > \overline{\mathbf{MAX\_NCC}_1}\right)$$
$$\begin{cases} \bullet\ \alpha = \max(0, \mathbf{NVERTEX}_{\texttt{INITIAL}} - \overline{\mathbf{MIN\_NCC}_2} - \overline{\mathbf{MIN\_NCC}_1}, \\ \bullet\ \beta = \max(1, \overline{\mathbf{MAX\_NCC}_2}) + \max(1, \overline{\mathbf{MAX\_NCC}_1}). \end{cases} \quad (3.203)$$

Proof. Similar to Proposition 133. □

Proposition 141.

$$\begin{array}{l}\texttt{vpartition} \wedge \texttt{consecutive\_loops\_are\_connected}: \\ \mathbf{MAX\_NCC}_1 \leq \max(\mathbf{MIN\_NCC}_1, \mathbf{NVERTEX}_{\texttt{INITIAL}} - \alpha), \texttt{with}: \\ \bullet\ \alpha = \mathbf{MIN\_NCC}_2 \cdot \max(0, \mathbf{NCC}_2 - 1) + \mathbf{MAX\_NCC}_2 + \\ \qquad \mathbf{MIN\_NCC}_1 + \mathbf{MIN\_NCC}_1 \cdot \max(0, \mathbf{NCC}_2 - 3)\end{array} \tag{3.204}$$

Proof. Similar to Proposition 134. □

Proposition 142.

$$\begin{array}{l}\texttt{apartition} \wedge \mathbf{arc\_gen} = \mathit{PATH} \wedge \mathbf{MIN\_NCC}_1 > 1 \wedge \mathbf{MIN\_NCC}_2 > 1: \\ \mathbf{NCC}_2 \leq (\mathbf{MAX\_NCC}_2 > 0) + \left\lfloor \dfrac{\alpha}{\beta} \right\rfloor + ((\alpha \bmod \beta) + 1 \geq \mathbf{MIN\_NCC}_2), \texttt{with}: \\ \begin{cases} \bullet\ \alpha = \max(0, \mathbf{NVERTEX}_{\texttt{INITIAL}} - \mathbf{MAX\_NCC}_1 - \mathbf{MAX\_NCC}_2 + 1), \\ \bullet\ \beta = \mathbf{MIN\_NCC}_1 + \mathbf{MIN\_NCC}_2 - 2. \end{cases}\end{array} \tag{3.205}$$

Proof. Similar to Proposition 135. □

Proposition 143.

$$\begin{array}{l}\texttt{apartition} \wedge \mathbf{arc\_gen} = \mathit{PATH} \wedge \mathbf{MIN\_NCC}_1 > 1 \wedge \mathbf{MIN\_NCC}_2 > 1: \\ \mathbf{NCC}_2 \geq (\mathbf{MIN\_NCC}_2 > 0) + \left\lfloor \dfrac{\alpha}{\beta} \right\rfloor + ((\alpha \bmod \beta) + 1 > \mathbf{MAX\_NCC}_1, \texttt{with}: \\ \begin{cases} \bullet\ \alpha = \max(0, \mathbf{NVERTEX}_{\texttt{INITIAL}} - \mathbf{MIN\_NCC}_1 - \mathbf{MIN\_NCC}_2 + 1), \\ \bullet\ \beta = \mathbf{MAX\_NCC}_1 + \mathbf{MAX\_NCC}_2 - 2. \end{cases}\end{array} \tag{3.206}$$

Proof. Similar to Proposition 136. □

Graph invariants relating six characteristics of two final graphs

$\boxed{\mathbf{MAX\_NCC}_1, \mathbf{MAX\_NCC}_2, \mathbf{MIN\_NCC}_1, \mathbf{MIN\_NCC}_2, \mathbf{NCC}_1, \mathbf{NCC}_2}$

Proposition 144.

$$\texttt{apartition} \wedge \mathbf{arc\_gen} = PATH \wedge \mathbf{NVERTEX}_{\texttt{INITIAL}} > 0 :$$
$$\alpha \cdot \mathbf{MIN\_NCC}_1 + \mathbf{MAX\_NCC}_1 +$$
$$\beta \cdot \mathbf{MIN\_NCC}_2 + \mathbf{MAX\_NCC}_2 \leq \mathbf{NVERTEX}_{\texttt{INITIAL}} + \mathbf{NCC}_1 + \mathbf{NCC}_2 - 1, \texttt{ with} :$$
$$\begin{cases} \bullet\ \alpha = \max(0, \mathbf{NCC}_1 - 1), \\ \bullet\ \beta = \max(0, \mathbf{NCC}_2 - 1). \end{cases} \tag{3.207}$$

Proof. Let $CC(G_1) = \{CC_a^1 : a \in [\mathbf{NCC}1]\}$ and $CC(G_2) = \{CC_a^2 : a \in [\mathbf{NCC}2]\}$ 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 $(A_i)_{i \in [\mathbf{NCC}_1 + \mathbf{NCC}_2]}$ 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 $i \in [\mathbf{NCC}_1 + \mathbf{NCC}_2 - 1]$.
By inclusion-exclusion principle, since $A_i \cap A_j = \emptyset$ whenever $j \neq i + 1$, we obtain $\mathbf{NVERTEX}_{\texttt{INITIAL}} = \Sigma_{a \in [\mathbf{NCC}1]} |CC_a^1| + \Sigma_{a \in [\mathbf{NCC}2]} |CC_a^2| - \Sigma_{i \in [\mathbf{NCC}1 + \mathbf{NCC}2 - 1]} |A_i \cap A_{i+1}|$. Since $|A_i \cap A_{i+1}|$ is equal to 1 for every well defined i, we obtain $\Sigma_{a \in [\mathbf{NCC}1]} |CC_a^1| + \Sigma_{a \in [\mathbf{NCC}2]} |CC_a^2| = \mathbf{NVERTEX}_{\texttt{INITIAL}} + \mathbf{NCC}1 + \mathbf{NCC}2 - 1$.
Since $\alpha \cdot \mathbf{MIN\_NCC}_1 + \mathbf{MAX\_NCC}_1 + \beta \cdot \mathbf{MIN\_NCC}_2 + \mathbf{MAX\_NCC}_2 \leq \Sigma_{a \in [\mathbf{NCC}1]} |CC_a^1| + \Sigma_{a \in [\mathbf{NCC}2]} |CC_a^2|$ the result follows. □

Proposition 145.

$$\texttt{apartition} \wedge \mathbf{arc\_gen} = PATH \wedge \mathbf{NVERTEX}_{\texttt{INITIAL}} > 0 :$$
$$\alpha \cdot \mathbf{MAX\_NCC}_1 + \mathbf{MIN\_NCC}_1 +$$
$$\beta \cdot \mathbf{MAX\_NCC}_2 + \mathbf{MIN\_NCC}_2 \geq \mathbf{NVERTEX}_{\texttt{INITIAL}} + \mathbf{NCC}_1 + \mathbf{NCC}_2 - 1, \texttt{ with} :$$
$$\begin{cases} \bullet\ \alpha = \max(0, \mathbf{NCC}_1 - 1), \\ \bullet\ \beta = \max(0, \mathbf{NCC}_2 - 1). \end{cases} \tag{3.208}$$

Proof. Similar to Proposition 144. □

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 `ORTHOTOPES` 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

Origin Inspired by [56].

Constraint all_differ_from_at_least_k_pos(K, VECTORS)

Type(s) VECTOR : collection(var − dvar)

Argument(s)

K	:	int
VECTORS	:	collection(vec − VECTOR)

Restriction(s)
required(VECTOR, var)
K ≥ 0
required(VECTORS, vec)
same_size(VECTORS, vec)

Purpose Enforce all pairs of distinct vectors of the VECTORS collection to differ from at least K positions.

Arc input(s) VECTORS

Arc generator $CLIQUE(\neq) \mapsto$ collection(vectors1, vectors2)

Arc arity 2

Arc constraint(s) differ_from_at_least_k_pos(K, vectors1.vec, vectors2.vec)

Graph property(ies) $\mathbf{NARC} = |\text{VECTORS}| * |\text{VECTORS}| - |\text{VECTORS}|$

Example

$$\texttt{all\_differ\_from\_at\_least\_k\_pos}\left(\begin{array}{l} 2, \left\{\begin{array}{l} \texttt{vec} - \left\{\begin{array}{l} \texttt{var} - 2, \\ \texttt{var} - 5, \\ \texttt{var} - 2, \\ \texttt{var} - 0 \end{array}\right\}, \\ \texttt{vec} - \left\{\begin{array}{l} \texttt{var} - 3, \\ \texttt{var} - 6, \\ \texttt{var} - 2, \\ \texttt{var} - 1 \end{array}\right\}, \\ \texttt{vec} - \left\{\begin{array}{l} \texttt{var} - 3, \\ \texttt{var} - 6, \\ \texttt{var} - 1, \\ \texttt{var} - 0 \end{array}\right\} \end{array}\right\} \end{array}\right)$$

The previous constraint holds since exactly $3 \cdot (3-1) = 6$ arc constraints hold, namely[1]:

- The first and second vectors differ from 3 positions which is greater than or equal to $\texttt{K} = 2$.

[1] Each item corresponds to two arc constraints.

- The first and third vectors differ from 3 positions which is greater than or equal to $\texttt{K} = 2$.
- The second and third vectors differ from 2 positions which is greater than or equal to $\texttt{K} = 2$.

Parts (A) and (B) of Figure 4.1 respectively show the initial and final graph. Since we use the $\mathbf{NARC}$ graph property, the arcs of the final graph are stressed in bold.

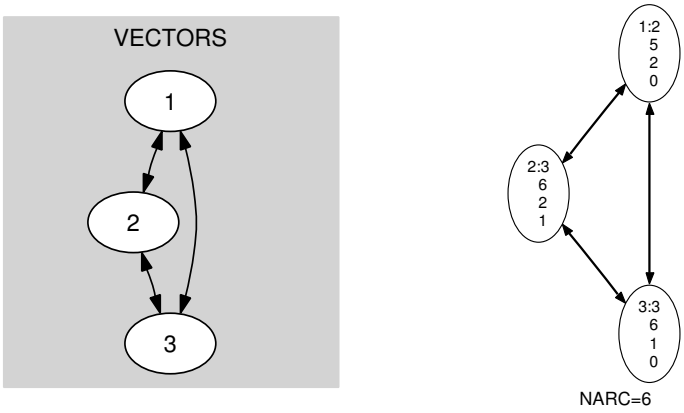


Figure 4.1: Initial and final graph of the `all_differ_from_at_least_k_pos` constraint

Graph model The arc constraint(s) field uses the `differ_from_at_least_k_pos` constraint defined in this catalog.

Signature Since we use the $CLIQUE(\neq)$ arc generator on the items of the `VECTORS` collection, the expression $|\texttt{VECTORS}| \cdot |\texttt{VECTORS}| - |\texttt{VECTORS}|$ corresponds to the maximum number of arcs of the final graph. Therefore we can rewrite the graph property $\mathbf{NARC} = |\texttt{VECTORS}| \cdot |\texttt{VECTORS}| - |\texttt{VECTORS}|$ to $\mathbf{NARC} \geq |\texttt{VECTORS}| \cdot |\texttt{VECTORS}| - |\texttt{VECTORS}|$. This leads to simplify $\underline{\mathbf{NARC}}$ to $\overline{\mathbf{NARC}}$.

See also `differ_from_at_least_k_pos`.

Key words decomposition, disequality, bioinformatics, vector, no_loop.

4.2 all_min_dist

Origin [57]

Constraint all_min_dist(MINDIST, VARIABLES)

Synonym(s) minimum_distance.

Argument(s)

MINDIST	:	int
VARIABLES	:	collection(var − dvar)

Restriction(s)

MINDIST > 0
required(VARIABLES, var)
VARIABLES.var ≥ 0

Purpose Enforce for each pair $(\texttt{var}_i, \texttt{var}_j)$ of distinct variables of the collection VARIABLES that $|\texttt{var}_i - \texttt{var}_j| \geq$ MINDIST.

Arc input(s) VARIABLES

Arc generator $CLIQUE(<) \mapsto$ collection(variables1, variables2)

Arc arity 2

Arc constraint(s) abs(variables1.var − variables2.var) $\geq$ MINDIST

Graph property(ies) $\mathbf{NARC} = |\texttt{VARIABLES}| * (|\texttt{VARIABLES}| - 1)/2$

Example all_min_dist ($2, \{\texttt{var} - 5, \texttt{var} - 1, \texttt{var} - 9, \texttt{var} - 3\}$)

Parts (A) and (B) of Figure 4.2 respectively show the initial and final graph. The all_min_dist constraint holds since all the arcs of the initial graph belong to the final graph: all the minimum distance constraints are satisfied.

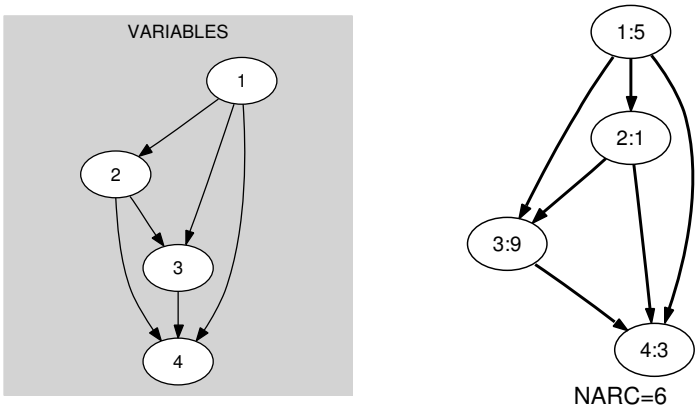
Graph model We generate a *clique* with a minimum distance constraint between each pair of distinct vertices and state that the number of arcs of the final graph should be equal to the number of arcs of the initial graph.

Usage The all_min_dist constraint was initially created for handling frequency allocation problems.

Remark The all_min_dist constraint can be modeled as a set of tasks which should not overlap. For each variable var of the VARIABLES collection we create a task t where var and MINDIST respectively correspond to the origin and the duration of t.

See also alldifferent, diffn.

Key words value constraint, decomposition, frequency allocation problem.



(A) **(B)**

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.

Arc input(s) VARIABLES

Arc generator $CLIQUE \mapsto$ collection(variables1, variables2)

Arc arity 2

Arc constraint(s) variables1.var = variables2.var

Graph property(ies) $\mathbf{MAX\_NSCC} \leq 1$

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 $\mathbf{MAX\_NSCC}$ 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 *clique* with an *equality* 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:

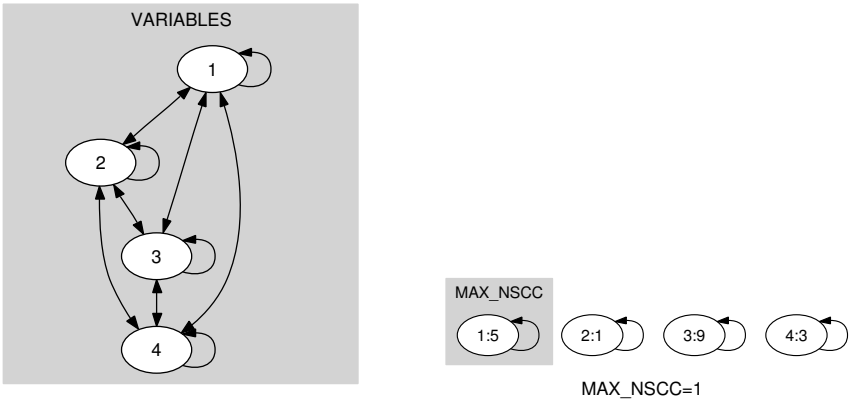


Figure 4.3: Initial and final graph of the `alldifferent` constraint

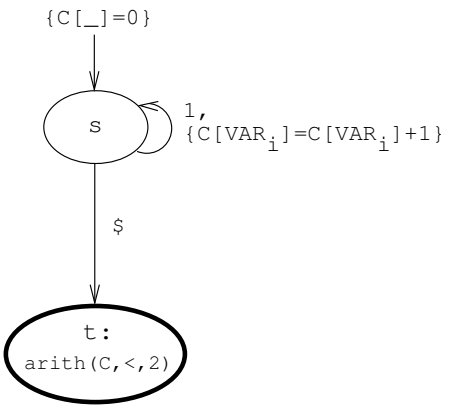


Figure 4.4: Automaton of the `alldifferent` constraint

- `alldifferent`$(X_1, X_2 + 1, \ldots, X_n + n - 1)$ for the upper-left to lower-right diagonals,
- `alldifferent`$(X_1, X_2, \ldots, X_n)$ for the lines,
- `alldifferent`$(X_1 + n - 1, X_2 + n - 2, \ldots, X_n)$ for the lower-right to upper-left diagonals.

They are respectively depicted by parts (A), (C) and (D) of Figure 4.5.

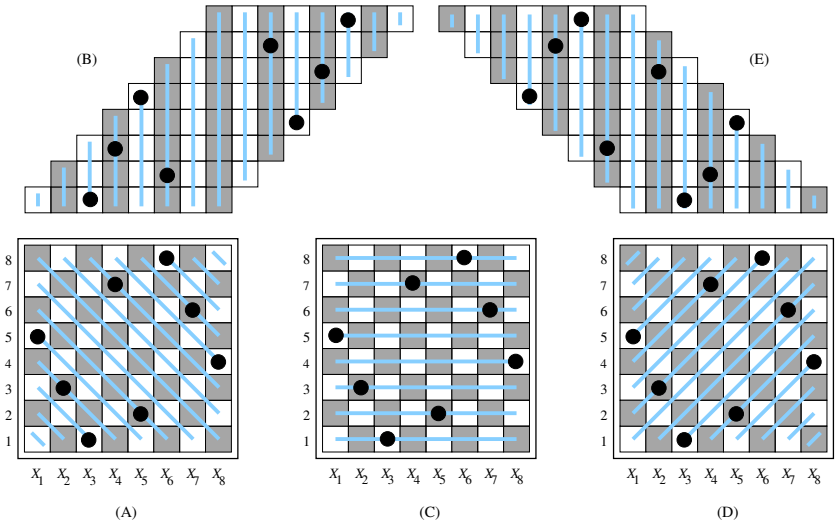


Figure 4.5: Upper-left to lower-right diagonals (A-B), lines (C) and lower-right to upper-left diagonals (D-E)

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.

Arc input(s) VARIABLES

Arc generator $CLIQUE \mapsto$ collection(variables1, variables2)

Arc arity 2

Arc constraint(s) eq_set(variables1.var, variables2.var)

Graph property(ies) $\mathbf{MAX\_NSCC} \leq 1$

Example alldifferent_between_sets $\left(\left\{ \begin{array}{l} \mathtt{var} - \{3,5\}, \\ \mathtt{var} - \emptyset, \\ \mathtt{var} - \{3\}, \\ \mathtt{var} - \{3,5,7\} \end{array} \right\} \right)$

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 alldifferent_between_sets holds since all the strongly connected components have at most one vertex.

Graph model We generate a *clique* with binary *set equalities* 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.

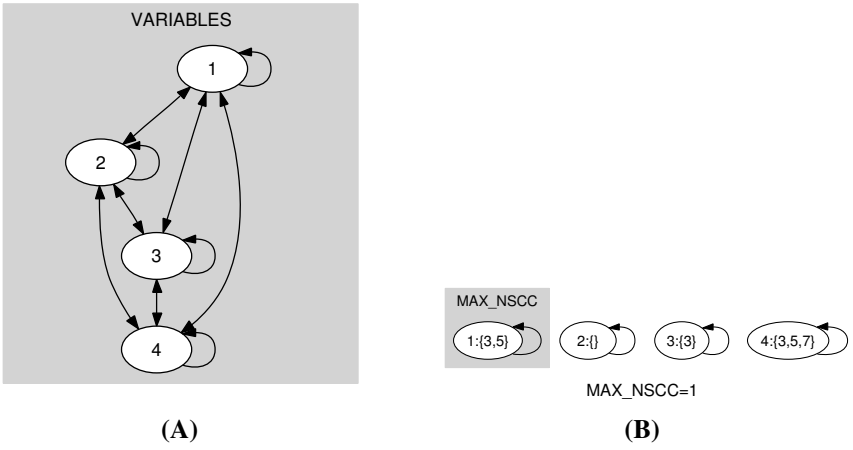


Figure 4.6: Initial and final graph of the `alldifferent_between_sets` constraint

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 $CLIQUE \mapsto$ `collection(variables1, variables2)`

Arc arity 2

Arc constraint(s)
- `variables1.var` $\neq 0$
- `variables1.var` $=$ `variables2.var`

Graph property(ies) $\mathbf{MAX\_NSCC} \leq 1$

Example

$$\texttt{alldifferent\_except\_0}\left(\left\{\begin{array}{l}\texttt{var}-5,\\ \texttt{var}-0,\\ \texttt{var}-1,\\ \texttt{var}-9,\\ \texttt{var}-0,\\ \texttt{var}-3\end{array}\right\}\right)$$

Parts (A) and (B) of Figure 4.7 respectively show the initial and final graph. Since we use the $\mathbf{MAX\_NSCC}$ 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.

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 $\texttt{VAR}_i$ of the collection `VARIABLES` corresponds a 0-1 signature variable $\texttt{S}_i$. The following signature constraint links $\texttt{VAR}_i$ and $\texttt{S}_i$: $\texttt{VAR}_i \neq 0 \Leftrightarrow \texttt{S}_i$. 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.

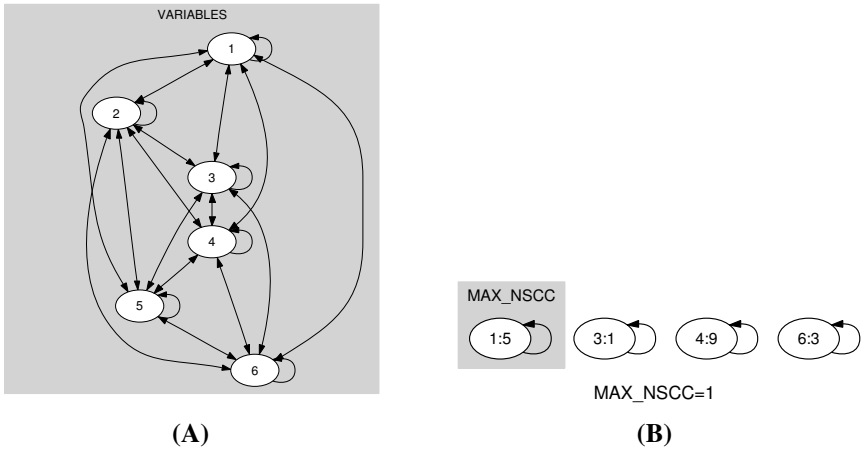


Figure 4.7: Initial and final graph of the `alldifferent_except_0` constraint

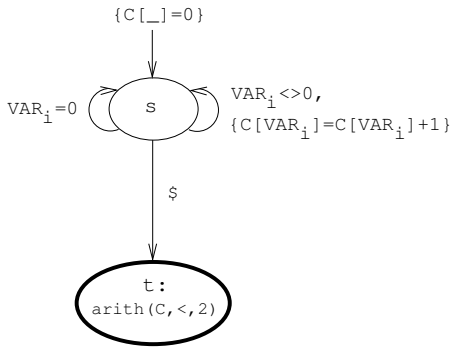


Figure 4.8: Automaton of the `alldifferent_except_0` constraint

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 `alldifferent`.

Constraint `alldifferent_interval(VARIABLES, SIZE_INTERVAL)`

Synonym(s) `alldiff_interval`, `alldistinct_interval`.

Argument(s)

`VARIABLES`	:	`collection(var − dvar)`
`SIZE_INTERVAL`	:	`int`

Restriction(s) `required(VARIABLES, var)`
$\texttt{SIZE\_INTERVAL} > 0$

Purpose Enforce all variables of the collection `VARIABLES` to belong to distinct intervals. The intervals are defined by $[\texttt{SIZE\_INTERVAL} \cdot k, \texttt{SIZE\_INTERVAL} \cdot k + \texttt{SIZE\_INTERVAL} - 1]$ where k is an integer.

Arc input(s) `VARIABLES`

Arc generator $CLIQUE \mapsto$ `collection(variables1, variables2)`

Arc arity 2

Arc constraint(s) `variables1.var/SIZE_INTERVAL = variables2.var/SIZE_INTERVAL`

Graph property(ies) $\mathbf{MAX\_NSCC} \leq 1$

Example `alldifferent_interval({var − 2, var − 3, var − 10}, 3)`

In the previous example, the second parameter `SIZE_INTERVAL` 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 `VARIABLES` take values that are respectively located within the three following distinct intervals $[0, 2]$, $[3, 5]$ and $[9, 11]$, the `alldifferent_interval` constraint holds. Parts (A) and (B) of Figure 4.9 respectively show the initial and final graph. Since we use the $\mathbf{MAX\_NSCC}$ graph property we show one of the largest strongly connected component of the final graph.

Graph model Similar to the `alldifferent` constraint, but we replace the binary *equality* constraint of the `alldifferent` constraint by the fact that two variables are respectively assigned to two values that belong to the same interval. We generate a *clique* with a *belong to the same interval* 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 4.10 depicts the automaton associated to the `alldifferent_interval` constraint. To each item of the collection `VARIABLES` corresponds a signature variable $\mathtt{S}_i$, which is equal to 1. For each interval $[\texttt{SIZE\_INTERVAL} \cdot k, \texttt{SIZE\_INTERVAL} \cdot k + \texttt{SIZE\_INTERVAL} - 1]$ 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.

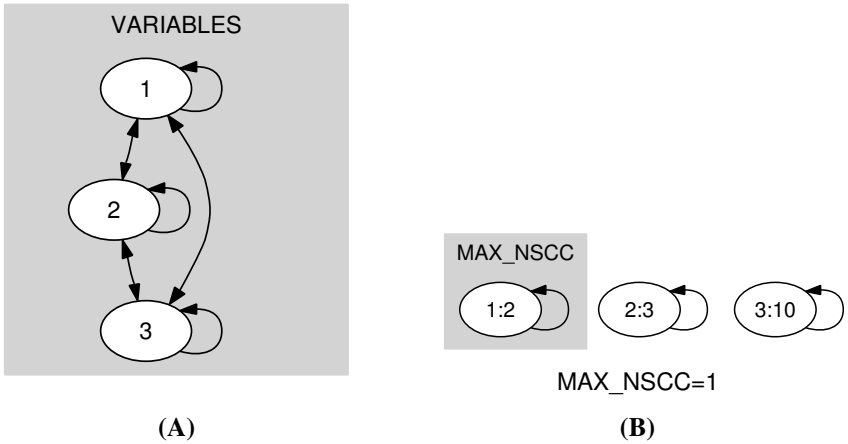


Figure 4.9: Initial and final graph of the `alldifferent_interval` constraint

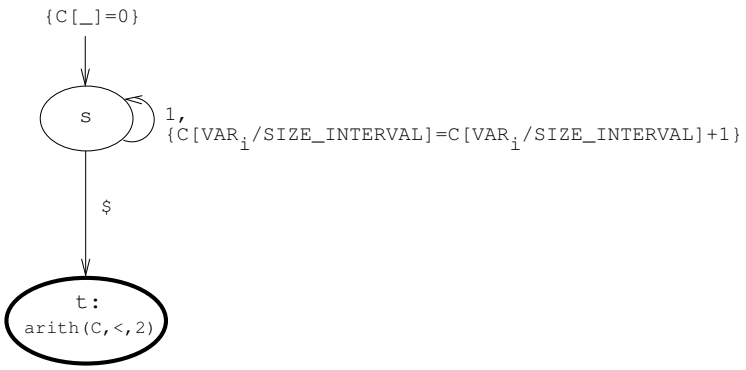


Figure 4.10: Automaton of the `alldifferent_interval` constraint

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 `alldifferent`.

Constraint `alldifferent_modulo(VARIABLES, M)`

Synonym(s) `alldiff_modulo`, `alldistinct_modulo`.

Argument(s)

`VARIABLES`	:	`collection(var − dvar)`
`M`	:	`int`

Restriction(s) `required(VARIABLES, var)`
$\texttt{M} \neq 0$
$\texttt{M} \geq |\texttt{VARIABLES}|$

Purpose Enforce all variables of the collection `VARIABLES` to have a distinct rest when divided by `M`.

Arc input(s) `VARIABLES`

Arc generator $CLIQUE \mapsto$ `collection(variables1, variables2)`

Arc arity 2

Arc constraint(s) $\texttt{variables1.var} \bmod \texttt{M} = \texttt{variables2.var} \bmod \texttt{M}$

Graph property(ies) $\mathbf{MAX\_NSCC} \leq 1$

Example $\texttt{alldifferent\_modulo}\left(\left\{\begin{array}{l}\texttt{var} - 25,\\ \texttt{var} - 1,\\ \texttt{var} - 14,\\ \texttt{var} - 3\end{array}\right\}, 5\right)$

The equivalences 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 `alldifferent_modulo` constraint holds. Parts (A) and (B) of Figure 4.11 respectively show the initial and final graph. Since we use the $\mathbf{MAX\_NSCC}$ graph property we show one of the largest strongly connected component of the final graph.

Graph model Exploit the same model used for the `alldifferent` constraint. We replace the binary *equality* constraint by an other equivalence relation depicted by the arc constraint. We generate a *clique* with a binary *equality modulo* `M` 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 `alldifferent_modulo` constraint. To each item of the collection `VARIABLES` corresponds a signature variable $\texttt{S}_i$, 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.

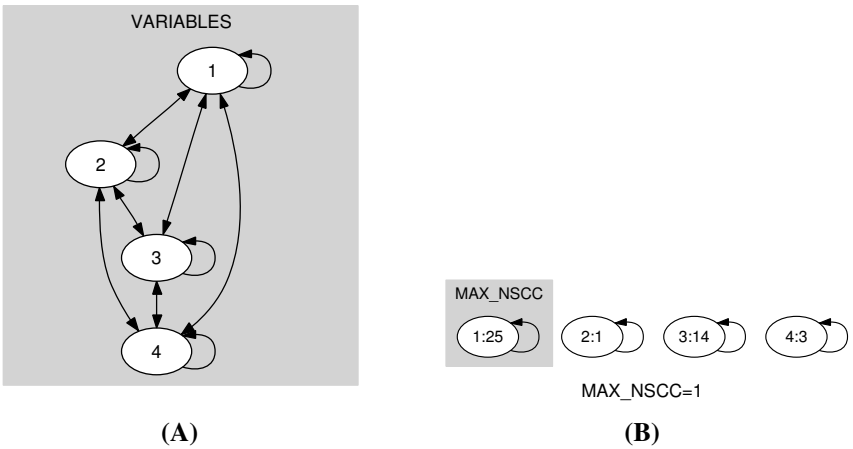


Figure 4.11: Initial and final graph of the `alldifferent_modulo` constraint

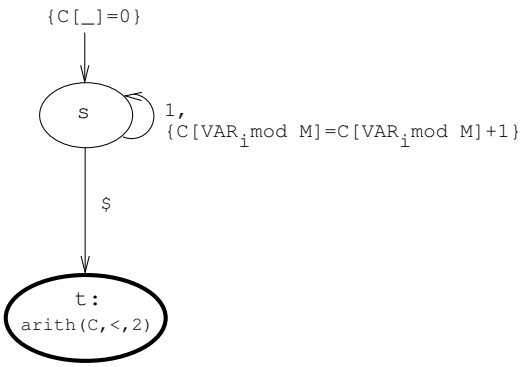


Figure 4.12: Automaton of the `alldifferent_modulo` constraint

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 $PRODUCT \mapsto$ collection(variables1, variables2)

Arc arity 2

Arc constraint(s) variables1.var = variables2.var

Graph property(ies) $\mathbf{MAX\_NCC} \leq 2$

Example

$$\texttt{alldifferent\_on\_intersection}\left(\left\{\begin{array}{l}\texttt{var}-5,\\ \texttt{var}-9,\\ \texttt{var}-1,\\ \texttt{var}-5\end{array}\right\},\left\{\begin{array}{l}\texttt{var}-2,\\ \texttt{var}-1,\\ \texttt{var}-6,\\ \texttt{var}-9,\\ \texttt{var}-6,\\ \texttt{var}-2\end{array}\right\}\right)$$

Parts (A) and (B) of Figure 4.13 respectively show the initial and final graph. Since we use the $\mathbf{MAX\_NCC}$ 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 $\texttt{VAR1}_i$ of the collection VARIABLES1 corresponds a signature variable $\texttt{S}_i$, which is equal to 0. To each variable $\texttt{VAR2}_i$ of the collection VARIABLES2 corresponds a signature variable $\texttt{S}_{i+|\texttt{VARIABLES1}|}$, 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.

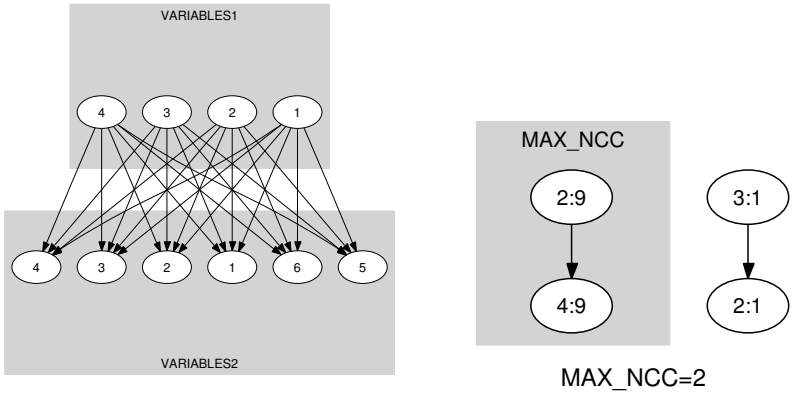


Figure 4.13: Initial and final graph of the `alldifferent_on_intersection` constraint

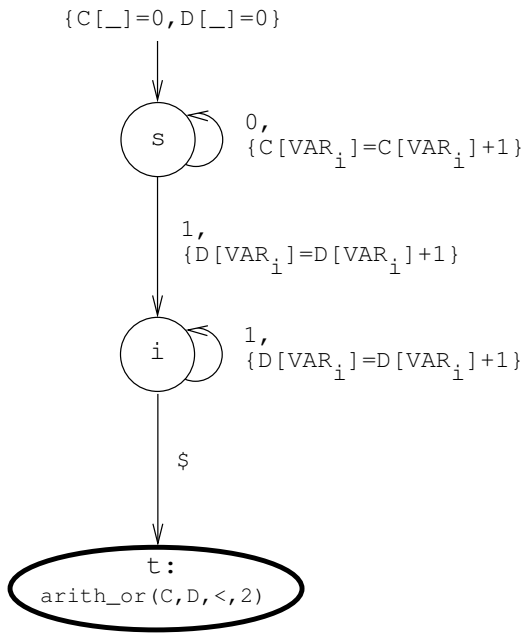


Figure 4.14: Automaton of the `alldifferent_on_intersection` constraint

4.9 alldifferent_partition

Origin Derived from `alldifferent`.

Constraint `alldifferent_partition(VARIABLES, PARTITIONS)`

Synonym(s) `alldiff_partition`, `alldistinct_partition`.

Type(s) `VALUES : collection(val − int)`

Argument(s)

`VARIABLES`	:	`collection(var − dvar)`
`PARTITIONS`	:	`collection(p − VALUES)`

Restriction(s)
`required(VALUES, val)`
`distinct(VALUES, val)`
$|\texttt{VARIABLES}| \leq |\texttt{PARTITIONS}|$
`required(VARIABLES, var)`
$|\texttt{PARTITIONS}| \geq 2$
`required(PARTITIONS, p)`

Purpose Enforce all variables of the collection `VARIABLES` to take values which belong to distinct partitions.

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) $\mathbf{MAX\_NSCC} \leq 1$

Example $\texttt{alldifferent\_partition}\left(\begin{array}{l}\{\texttt{var}-6, \texttt{var}-3, \texttt{var}-4\},\\ \left\{\begin{array}{l}\texttt{p}-\{\texttt{val}-1, \texttt{val}-3\},\\ \texttt{p}-\{\texttt{val}-4\},\\ \texttt{p}-\{\texttt{val}-2, \texttt{val}-6\}\end{array}\right\}\end{array}\right)$

Since all variables take values that are located within distinct partitions the `alldifferent_partition` constraint holds. Parts (A) and (B) of Figure 4.15 respectively show the initial and final graph. Since we use the $\mathbf{MAX\_NSCC}$ graph property we show one of the largest strongly connected component of the final graph.

Graph model Similar to the `alldifferent` constraint, but we replace the binary *equality* constraint of the `alldifferent` constraint by the fact that two variables are respectively assigned to two values that belong to the same partition. We generate a *clique* with a `in_same_partition` 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 `alldifferent`, `in_same_partition`.

Key words value constraint, partition, all different, one_succ.

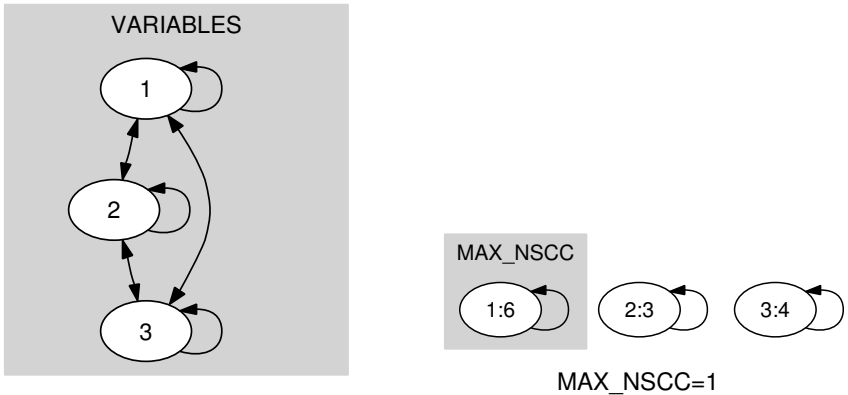


Figure 4.15: Initial and final graph of the `alldifferent_partition` constraint

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 $\leq$ |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 $(1 \leq i \leq$ |VARIABLES1|$)$ that hold.

Arc input(s) VARIABLES1 VARIABLES2

Arc generator $PRODUCT(CLIQUE, LOOP, =) \mapsto$ collection(variables1, variables2)

Arc arity 2

Arc constraint(s) variables1.var = variables2.var

Graph property(ies)
- **MAX_NSCC** ≤ 1
- **NARC_NO_LOOP** = NSAME

Example

$$\texttt{alldifferent\_same\_value}\left(\begin{array}{l} 2, \left\{\begin{array}{l}\texttt{var}-7,\\ \texttt{var}-3,\\ \texttt{var}-1,\\ \texttt{var}-5\end{array}\right\},\\ \left\{\begin{array}{l}\texttt{var}-1,\\ \texttt{var}-3,\\ \texttt{var}-1,\\ \texttt{var}-7\end{array}\right\}\end{array}\right)$$

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.

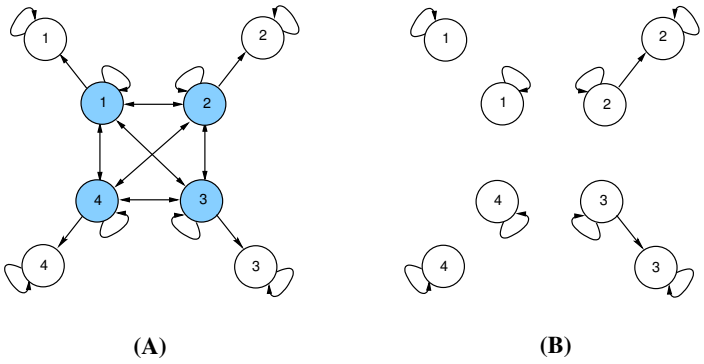
Automaton Figure 4.17 depicts the automaton associated to the alldifferent_same_value constraint. Let $\texttt{VAR1}_i$ and $\texttt{VAR2}_i$ respectively denote the i^{th} variables of the VARIABLES1 and VARIABLES2 collections. To each pair of variables $(\texttt{VAR1}_i, \texttt{VAR2}_i)$ corresponds a signature variable $\texttt{S}_i$. The following signature constraint links $\texttt{VAR1}_i$, $\texttt{VAR2}_i$ and $\texttt{S}_i$: $\texttt{VAR1}_i = \texttt{VAR2}_i \Leftrightarrow \texttt{S}_i$.

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 mesures 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.



(A) **(B)**

Figure 4.16: Initial and final graph of the alldifferent_same_value constraint

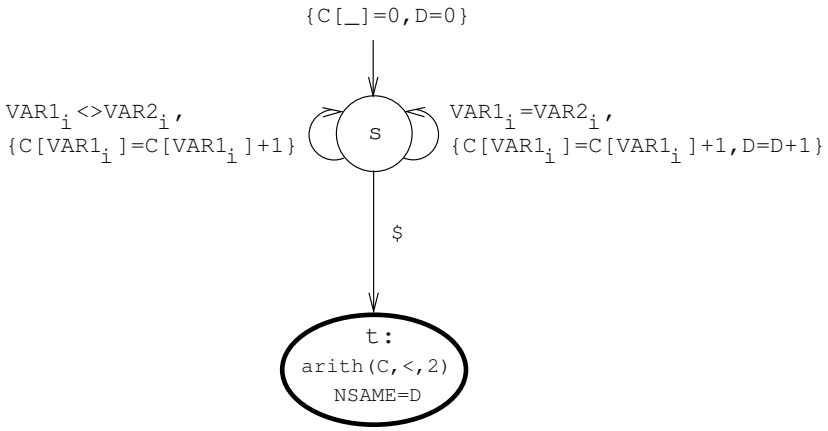


Figure 4.17: Automaton of the `alldifferent_same_value` constraint

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 $\texttt{allperm}\left(\left\{\begin{array}{l}\texttt{vec} - \{\texttt{var}-1, \texttt{var}-2, \texttt{var}-3\},\\ \texttt{vec} - \{\texttt{var}-3, \texttt{var}-1, \texttt{var}-2\}\end{array}\right\}\right)$

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. $\langle 1, 2, 3\rangle$, $\langle 1, 3, 2\rangle$, $\langle 2, 1, 3\rangle$, $\langle 2, 3, 1\rangle$, $\langle 3, 1, 2\rangle$, $\langle 3, 2, 1\rangle$).

Usage A *symmetry-breaking* constraint.

See also lex2, lex_lesseq.

Key words predefined constraint, order constraint, matrix, matrix model, symmetry, lexicographic order.

4.12 among

Origin [37]

Constraint among(NVAR, VARIABLES, VALUES)

Argument(s)

NVAR	:	dvar
VARIABLES	:	collection(var − dvar)
VALUES	:	collection(val − int)

Restriction(s)

NVAR ≥ 0
NVAR $\leq |$VARIABLES$|$
required(VARIABLES, var)
required(VALUES, val)
distinct(VALUES, val)

Purpose NVAR is the number of variables of the collection VARIABLES which take their value in VALUES.

Arc input(s) VARIABLES

Arc generator $SELF \mapsto$ collection(variables)

Arc arity 1

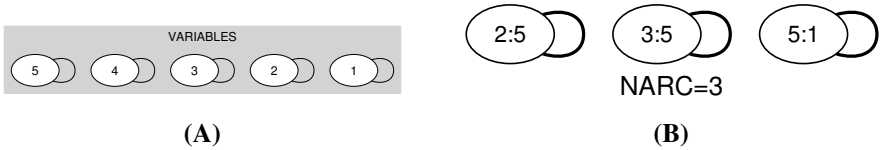
Arc constraint(s) in(variables.var, VALUES)

Graph property(ies) $\mathbf{NARC} =$ NVAR

Example

$$\texttt{among}\left(\begin{array}{c} 3, \left\{\begin{array}{l}\texttt{var}-4,\\ \texttt{var}-5,\\ \texttt{var}-5,\\ \texttt{var}-4,\\ \texttt{var}-1\end{array}\right\},\\ \{\texttt{val}-1, \texttt{val}-5, \texttt{val}-8\}\end{array}\right)$$

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.



(A) **(B)**

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 $\texttt{VAR}_i$ of the collection VARIABLES corresponds a 0-1 signature variable $\texttt{S}_i$. The following signature constraint links $\texttt{VAR}_i$ and $\texttt{S}_i$: $\texttt{VAR}_i \in \texttt{VALUES} \Leftrightarrow \texttt{S}_i$. The automaton counts the number of variables of the VARIABLES collection which take their value in VALUES and finally assigns this number to NVAR.

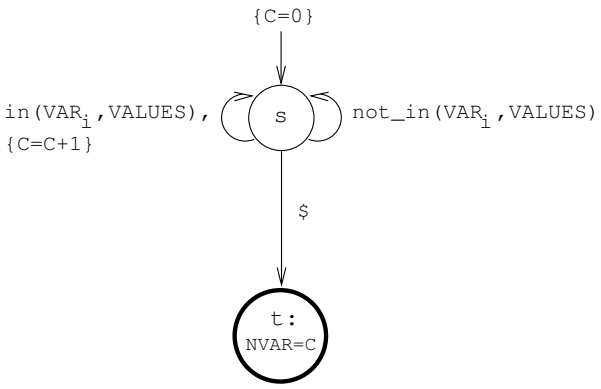


Figure 4.19: Automaton of the among constraint

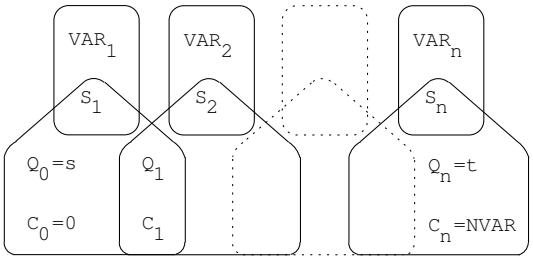


Figure 4.20: Hypergraph of the reformulation corresponding to the automaton of the among constraint

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 nvalue.

Constraint among_diff_0(NVAR, VARIABLES)

Argument(s)

NVAR	:	dvar
VARIABLES	:	collection(var − dvar)

Restriction(s)

NVAR ≥ 0
NVAR $\leq |$VARIABLES$|$
required(VARIABLES, var)

Purpose NVAR is the number of variables of the collection VARIABLES which take a value different from 0.

Arc input(s) VARIABLES

Arc generator $SELF \mapsto$ collection(variables)

Arc arity 1

Arc constraint(s) variables.var $\neq 0$

Graph property(ies) $\mathbf{NARC} =$ NVAR

Example

$$\texttt{among\_diff\_0}\left(\begin{array}{l} 3, \left\{\begin{array}{l}\texttt{var}-0,\\ \texttt{var}-5,\\ \texttt{var}-5,\\ \texttt{var}-0,\\ \texttt{var}-1\end{array}\right\}\end{array}\right)$$

Parts (A) and (B) of Figure 4.21 respectively show the initial and final graph. Since we use the $\mathbf{NARC}$ graph property, the unary arcs of the final graph are stressed in bold.

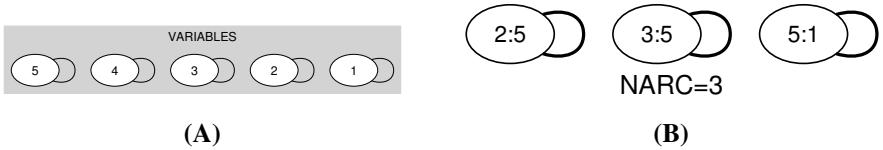


Figure 4.21: Initial and final graph of the among_diff_0 constraint

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.

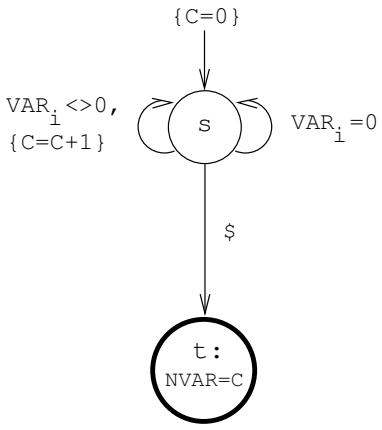


Figure 4.22: Automaton of the among_diff_0 constraint

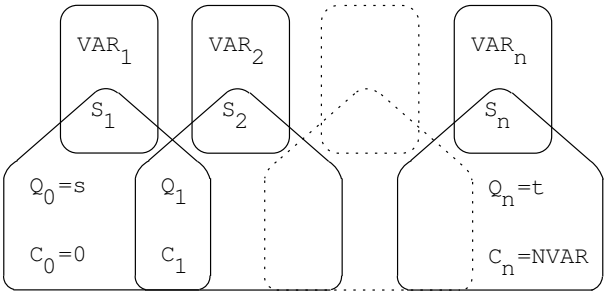


Figure 4.23: Hypergraph of the reformulation corresponding to the automaton of the among_diff_0 constraint

Automaton Figure 4.22 depicts the automaton associated to the among_diff_0 constraint. To each variable $\texttt{VAR}_i$ of the collection VARIABLES corresponds a 0-1 signature variable $\texttt{S}_i$. The following signature constraint links $\texttt{VAR}_i$ and $\texttt{S}_i$: $\texttt{VAR}_i \neq 0 \Leftrightarrow \texttt{S}_i$. 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).

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 $SELF \mapsto$ collection(variables)

Arc arity 1

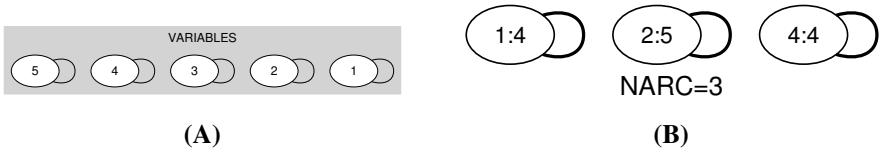
Arc constraint(s)
- LOW ≤ variables.var
- variables.var ≤ UP

Graph property(ies) **NARC** = NVAR

Example

$$\texttt{among\_interval}\left(\begin{array}{c} 3, \left\{\begin{array}{l} \texttt{var}-4,\\ \texttt{var}-5,\\ \texttt{var}-8,\\ \texttt{var}-4,\\ \texttt{var}-1 \end{array}\right\}, 3, 5 \end{array}\right)$$

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.



(A) **(B)**

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 $\texttt{VAR}_i$ of the collection VARIABLES corresponds a 0-1 signature variable $\texttt{S}_i$. The following signature constraint links $\texttt{VAR}_i$ and $\texttt{S}_i$: $\texttt{LOW} \leq \texttt{VAR}_i \wedge \texttt{VAR}_i \leq \texttt{UP} \Leftrightarrow \texttt{S}_i$. The automaton counts the number of variables of the VARIABLES collection which take their value in $[\texttt{LOW}, \texttt{UP}]$ and finally assigns this number to NVAR.

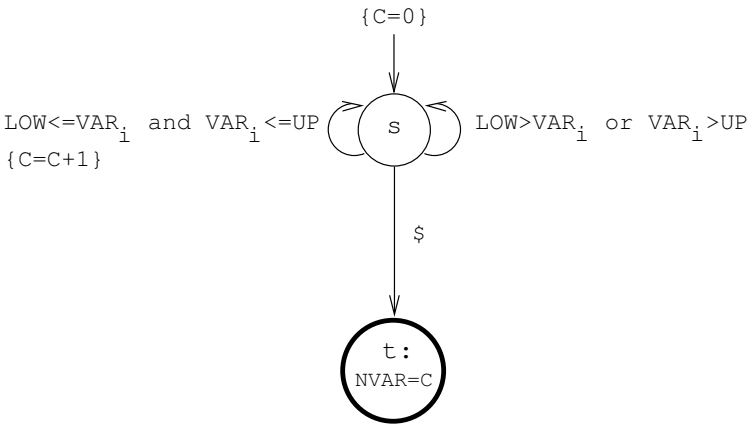


Figure 4.25: Automaton of the among_interval constraint

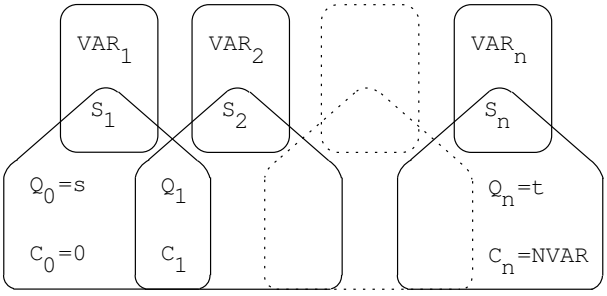


Figure 4.26: Hypergraph of the reformulation corresponding to the automaton of the among_interval constraint

Remark By giving explicitly all values of the interval $[\texttt{LOW}, \texttt{UP}]$ the among_interval constraint can be modelled with the among constraint. However when $\texttt{LOW} - \texttt{UP} + 1$ 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 ≥ 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 $PRODUCT \mapsto$ collection(variables, values)

Arc arity 2

Arc constraint(s) variables.var = values.val

Graph property(ies)
- **NARC** $\geq$ LOW
- **NARC** $\leq$ UP

Example among_low_up$\left(\begin{array}{l} 1, 2, \{\mathtt{var}-9, \mathtt{var}-2, \mathtt{var}-4, \mathtt{var}-5\}, \\ \left\{\begin{array}{l} \mathtt{val}-0, \\ \mathtt{val}-2, \\ \mathtt{val}-4, \\ \mathtt{val}-6, \\ \mathtt{val}-8 \end{array}\right\} \end{array}\right)$

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.

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 $\texttt{VAR}_i$ of the collection VARIABLES corresponds a 0-1 signature variable $\texttt{S}_i$. The following signature constraint links $\texttt{VAR}_i$ and $\texttt{S}_i$: $\texttt{VAR}_i \in \texttt{VALUES} \Leftrightarrow \texttt{S}_i$. 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 $[\texttt{LOW}, \texttt{UP}]$.

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.

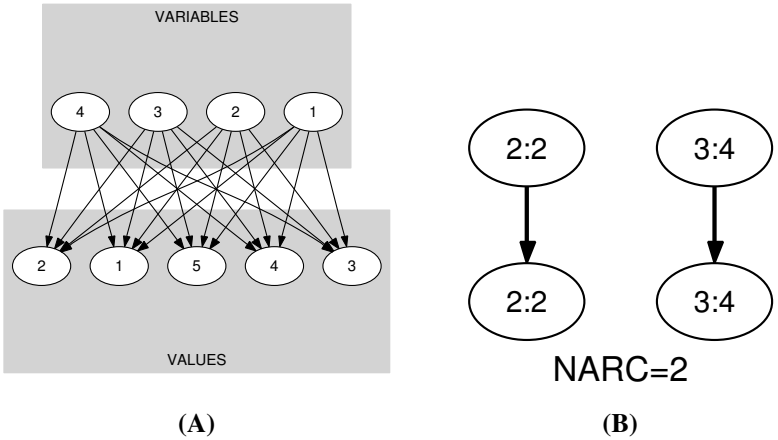


Figure 4.27: Initial and final graph of the among_low_up constraint

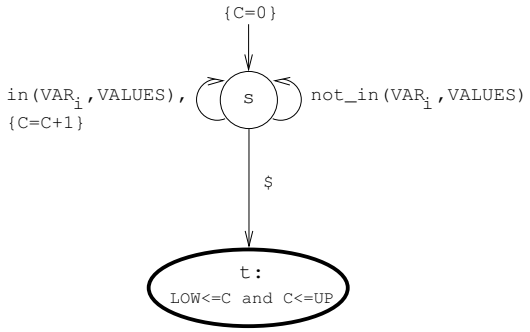


Figure 4.28: Automaton of the among_low_up constraint

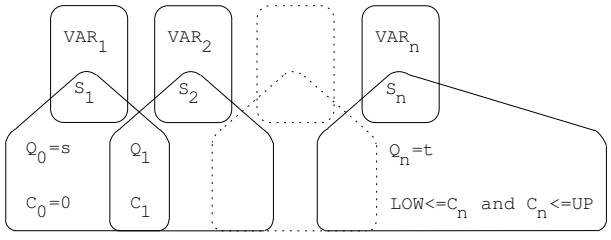


Figure 4.29: Hypergraph of the reformulation corresponding to the automaton of the among_low_up constraint

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 $\leq |$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

$$\text{among\_modulo}\left(3, \left\{\begin{array}{l}\text{var}-4,\\ \text{var}-5,\\ \text{var}-8,\\ \text{var}-4,\\ \text{var}-1\end{array}\right\}, 0, 2\right)$$

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.

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 4.31 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$_i$.

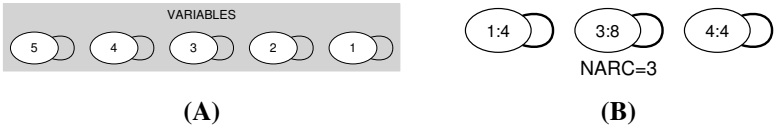


Figure 4.30: Initial and final graph of the among_modulo constraint

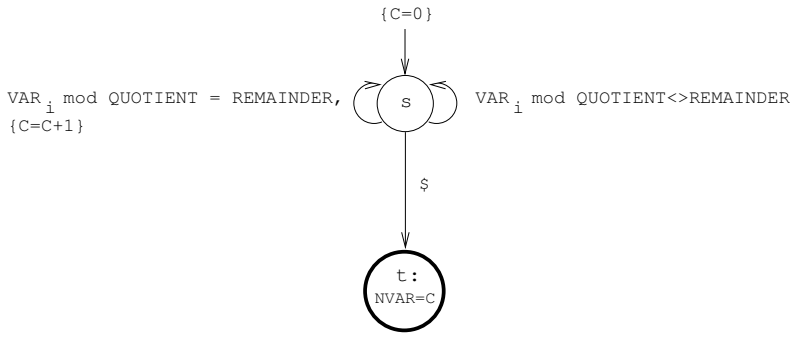


Figure 4.31: Automaton of the among_modulo constraint

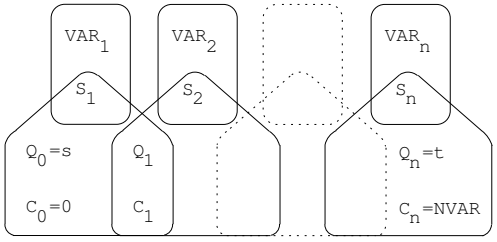


Figure 4.32: Hypergraph of the reformulation corresponding to the automaton of the among_modulo constraint

Remark By giving explicitly all values v which satisfy the equality $v \bmod \texttt{QUOTIENT} = \texttt{REMAINDER}$ 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).

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 ≥ 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) $\mathbf{NARC} = |$VARIABLES$| -$ SEQ $+ 1$

Example

$$\text{among\_seq}\left(\begin{array}{l} \left\{\begin{array}{l}\text{var}-9,\\ \text{var}-2,\\ \text{var}-4,\\ \text{var}-5,\\ \text{var}-5,\\ \text{var}-7,\\ \text{var}-2\end{array}\right\},\\ 1,2,4,\\ \left\{\begin{array}{l}\text{val}-0,\\ \text{val}-2,\\ \text{val}-4,\\ \text{val}-6,\\ \text{val}-8\end{array}\right\} \end{array}\right)$$

The previous constraint holds since the different sequences of 4 consecutive variables contains respectively 2, 2, 1 and 1 even numbers.

Graph model A constraint on sliding sequences of consecutives 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 $PATH$. The constraint associated to an arc corresponds to the among_low_up constraint defined at an other entry of this catalog.

Signature Since we use the $PATH$ arc generator with an arity of SEQ on the items of the VARIABLES collection, the expression $|\texttt{VARIABLES}| - \texttt{SEQ} + 1$ corresponds to the maximum number of arcs of the final graph. Therefore we can rewrite the graph property $\mathbf{NARC} = |\texttt{VARIABLES}| - \texttt{SEQ} + 1$ to $\mathbf{NARC} \geq |\texttt{VARIABLES}| - \texttt{SEQ} + 1$ and simplify $\underline{\mathbf{NARC}}$ to $\overline{\mathbf{NARC}}$.

Algorithm [65].

See also among, among_low_up.

Key words decomposition, sliding sequence constraint, sequence, hypergraph.

4.18 arith

Origin Used in the definition of several automata

Constraint arith(VARIABLES, RELOP, VALUE)

Argument(s)

VARIABLES	:	collection(var − dvar)
RELOP	:	atom
VALUE	:	int

Restriction(s) required(VARIABLES, var)
RELOP $\in [=, \neq, <, \geq, >, \leq]$

Purpose Enforce for all variables var of the VARIABLES collection to have var RELOP VALUE.

Arc input(s) VARIABLES

Arc generator $SELF \mapsto$ collection(variables)

Arc arity 1

Arc constraint(s) variables.var RELOP VALUE

Graph property(ies) $\mathbf{NARC} = |$VARIABLES$|$

Example $\texttt{arith}\left(\left\{\begin{array}{l}\texttt{var}-4,\\ \texttt{var}-5,\\ \texttt{var}-7,\\ \texttt{var}-4,\\ \texttt{var}-5\end{array}\right\}, <, 9\right)$

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.

(A) **(B)**

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$_i$: VAR$_i$ RELOP VALUE $\Leftrightarrow$ S$_i$. The automaton enforces for each variable VAR$_i$ the condition VAR$_i$ RELOP VALUE.

Used in arith_sliding.

See also among, count.

Key words decomposition, value constraint, domain definition, automaton, automaton without counters.

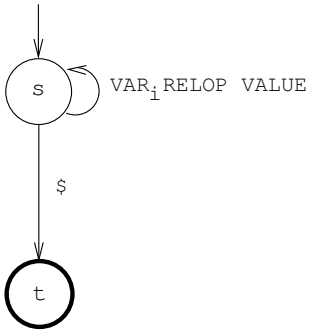


Figure 4.34: Automaton of the arith constraint

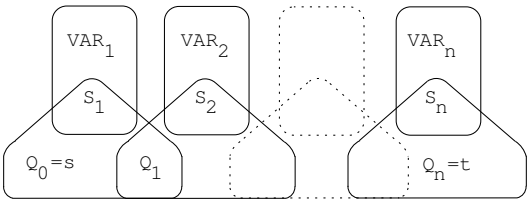


Figure 4.35: Hypergraph of the reformulation corresponding to the automaton of the `arith` constraint

4.19 arith_or

Origin Used in the definition of several automata

Constraint arith_or(VARIABLES1, VARIABLES2, RELOP, VALUE)

Argument(s)

VARIABLES1	:	collection(var − dvar)
VARIABLES2	:	collection(var − dvar)
RELOP	:	atom
VALUE	:	int

Restriction(s)
required(VARIABLES1, var)
required(VARIABLES2, var)
|VARIABLES1| = |VARIABLES2|
RELOP ∈ [=, ≠, <, ≥, >, ≤]

Purpose Enforce for all pairs of variables $\texttt{var1}_i, \texttt{var2}_i$ of the VARIABLES1 and VARIABLES2 collections to have $\texttt{var1}_i$ RELOP VALUE ∨ $\texttt{var2}_i$ RELOP VALUE.

Arc input(s) VARIABLES1 VARIABLES2

Arc generator $PRODUCT(=) \mapsto$ collection(variables1, variables2)

Arc arity 2

Arc constraint(s) variables1.var RELOP VALUE ∨ variables2.var RELOP VALUE

Graph property(ies) $\mathbf{NARC} = |\texttt{VARIABLES1}|$

Example

$$\texttt{arith\_or}\left(\left\{\begin{array}{l}\texttt{var}-0,\\ \texttt{var}-1,\\ \texttt{var}-0,\\ \texttt{var}-0,\\ \texttt{var}-1\end{array}\right\},\left\{\begin{array}{l}\texttt{var}-0,\\ \texttt{var}-0,\\ \texttt{var}-0,\\ \texttt{var}-1,\\ \texttt{var}-0\end{array}\right\},=,0\right)$$

The constraint holds since for all pairs of variables $\texttt{var1}_i, \texttt{var2}_i$ 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 $\texttt{VAR1}_i$ and $\texttt{VAR2}_i$ be the i^{th} variables of the VARIABLES1 and VARIABLES2 collections. To each pair of variables $(\texttt{VAR1}_i, \texttt{VAR2}_i)$ corresponds a signature variable $\texttt{S}_i$. The following signature constraint links $\texttt{VAR1}_i$, $\texttt{VAR2}_i$ and $\texttt{S}_i$: $\texttt{VAR1}_i$ RELOP VALUE ∨ $\texttt{VAR2}_i$ RELOP VALUE ⇔ $\texttt{S}_i$. The automaton enforces for each pair of variables $\texttt{VAR1}_i$,$\texttt{VAR2}_i$ the condition $\texttt{VAR1}_i$ RELOP VALUE ∨ $\texttt{VAR2}_i$ RELOP VALUE.

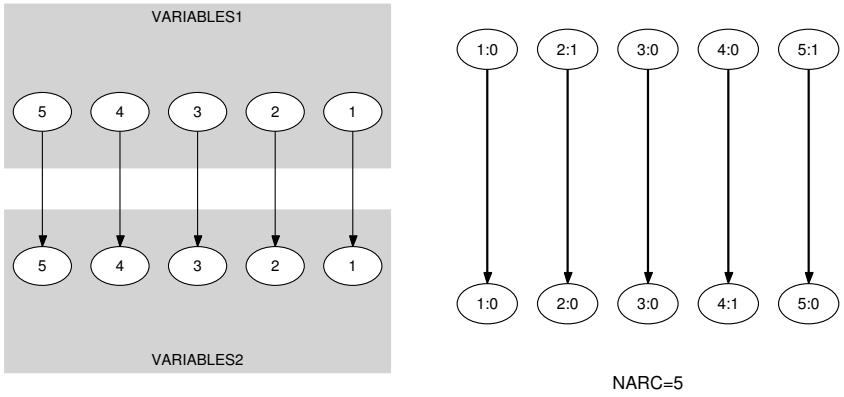


Figure 4.36: Initial and final graph of the `arith_or` constraint

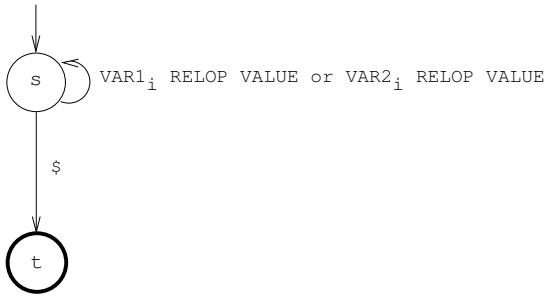


Figure 4.37: Automaton of the `arith_or` constraint

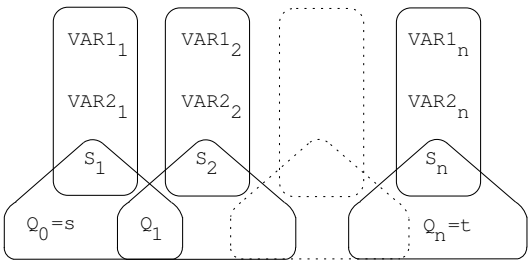


Figure 4.38: Hypergraph of the reformulation corresponding to the automaton of the `arith_or` constraint

See also `arith`.

Key words decomposition, value constraint, automaton, automaton without counters, acyclic, bipartite, no_loop.

4.20 arith_sliding

Origin Used in the definition of some automaton

Constraint arith_sliding(VARIABLES, RELOP, VALUE)

Argument(s)

VARIABLES	:	collection(var − dvar)
RELOP	:	atom
VALUE	:	int

Restriction(s) required(VARIABLES, var)
RELOP $\in [=, \neq, <, \geq, >, \leq]$

Purpose Enforce for all sequences of variables $\mathtt{var}_1, \mathtt{var}_2, \ldots, \mathtt{var}_i$ of the VARIABLES collection to have $(\mathtt{var}_1 + \mathtt{var}_2 + \ldots + \mathtt{var}_i)$ RELOP VALUE.

Arc input(s) VARIABLES

Arc generator $PATH\_1 \mapsto$ collection

Arc arity $*$

Arc constraint(s) arith(collection, RELOP, VALUE)

Graph property(ies) $\mathbf{NARC} = |\mathtt{VARIABLES}|$

Example

$$\mathtt{arith\_sliding}\left(\left\{\begin{array}{l}\mathtt{var}-0,\\ \mathtt{var}-0,\\ \mathtt{var}-1,\\ \mathtt{var}-2,\\ \mathtt{var}-0,\\ \mathtt{var}-0,\\ \mathtt{var}--3\end{array}\right\}, <, 4\right)$$

The previous constraint holds since all the following seven inequalities hold:

- $0 < 4$,
- $0 + 0 < 4$,
- $0 + 0 + 1 < 4$,
- $0 + 0 + 1 + 2 < 4$,
- $0 + 0 + 1 + 2 + 0 < 4$,
- $0 + 0 + 1 + 2 + 0 + 0 < 4$,
- $0 + 0 + 1 + 2 + 0 + 0 - 3 < 4$.

Automaton Figure 4.39 depicts the automaton associated to the arith_sliding constraint. To each item of the collection VARIABLES corresponds a signature variable $\mathtt{S}_i$, which is equal to 0.

See also arith, cumulative.

Key words decomposition, sliding sequence constraint, sequence, hypergraph, automaton, automaton with counters.

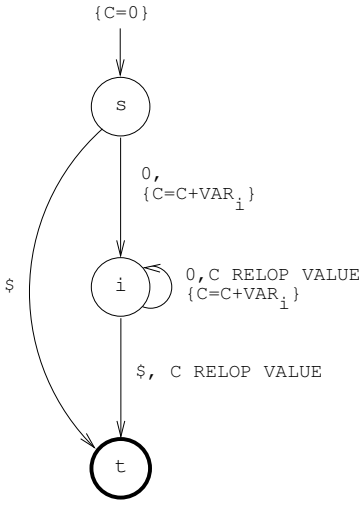


Figure 4.39: Automaton of the `arith_sliding` constraint

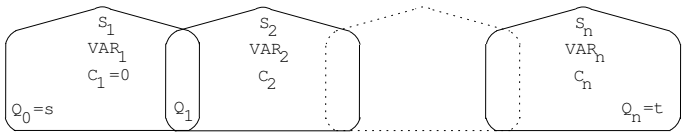


Figure 4.40: Hypergraph of the reformulation corresponding to the automaton of the `arith_sliding` constraint

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

$$\text{SUCC} \mapsto \left[\begin{array}{l} \texttt{source}, \\ \texttt{variables} - \texttt{col}\left(\begin{array}{l} \texttt{VARIABLES} - \texttt{collection}(\texttt{var} - \texttt{dvar}), \\ [\texttt{item}(\texttt{var} - \texttt{ITEMS.colour})] \end{array} \right) \end{array} \right]$$

Constraint(s) on sets counts(VALUES, variables, RELOP, LIMIT)

Example

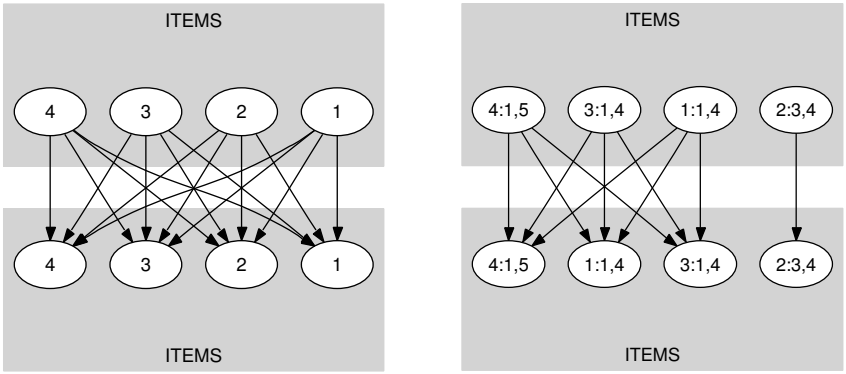
$$\texttt{assign\_and\_counts}\left(\begin{array}{l} \{\texttt{val} - 4\}, \\ \left\{ \begin{array}{ll} \texttt{bin} - 1 & \texttt{colour} - 4, \\ \texttt{bin} - 3 & \texttt{colour} - 4, \\ \texttt{bin} - 1 & \texttt{colour} - 4, \\ \texttt{bin} - 1 & \texttt{colour} - 5 \end{array} \right\}, \leq, 2 \end{array} \right)$$

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.



(A) **(B)**

Figure 4.41: Initial and final graph of the `assign_and_counts` constraint

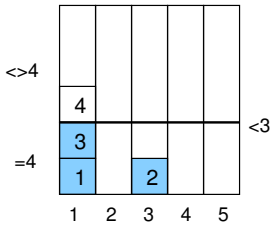


Figure 4.42: Assignment of the items to the bins

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 $\texttt{COLOUR}_i$ of the collection `ITEMS` corresponds a 0-1 signature variable $\texttt{S}_i$. The following signature constraint links $\texttt{COLOUR}_i$ and $\texttt{S}_i$: $\texttt{COLOUR}_i \in \texttt{COLOURS} \Leftrightarrow \texttt{S}_i$. 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 `RELOP LIMIT`.

Usage Some persons have pointed out that it is impossible to use constraints such as `among`, `atleast`, `atmost`, `count`, or `global_cardinality` if the set of variables is not initially known. However, this is for instance required in practice for some timetabling problems.

See also `count`, `counts`.

Key words assignment, coloured, automaton, automaton with array of counters, derived collection.

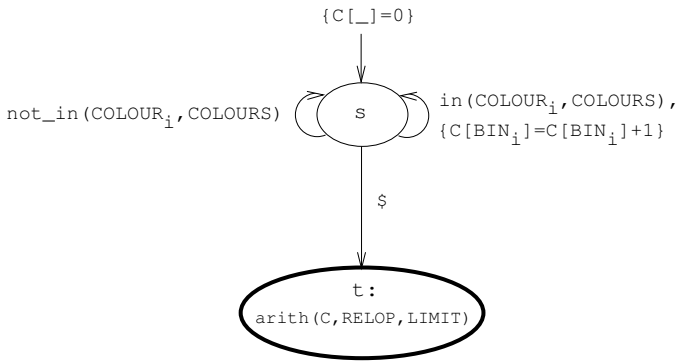


Figure 4.43: Automaton of the `assign_and_counts` constraint

4.22 assign_and_nvalues

Origin Derived from assign_and_counts and nvalues.

Constraint assign_and_nvalues(ITEMS, RELOP, LIMIT)

Argument(s)

ITEMS	:	collection(bin − dvar, value − dvar)
RELOP	:	atom
LIMIT	:	dvar

Restriction(s) required(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 $PRODUCT \mapsto$ collection(items1, items2)

Arc arity 2

Arc constraint(s) items1.bin = items2.bin

Sets
$$\text{SUCC} \mapsto \left[\begin{array}{l} \text{source}, \\ \text{variables} - \text{col}(\text{VARIABLES} - \text{collection}(\text{var} - \text{dvar}), [\text{item}(\text{var} - \text{ITEMS.value})]) \end{array}\right]$$

Constraint(s) on sets nvalues(variables, RELOP, LIMIT)

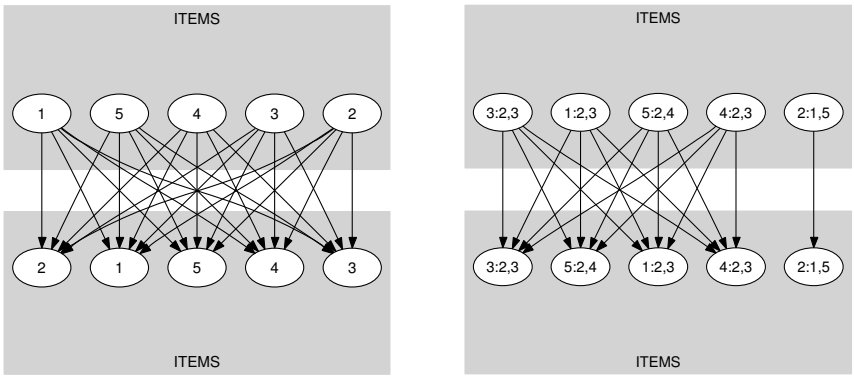
Example
$$\text{assign\_and\_nvalues}\left(\left\{\begin{array}{ll} \text{bin} - 2 & \text{value} - 3, \\ \text{bin} - 1 & \text{value} - 5, \\ \text{bin} - 2 & \text{value} - 3, \\ \text{bin} - 2 & \text{value} - 3, \\ \text{bin} - 2 & \text{value} - 4 \end{array}\right\}, \leq, 2\right)$$

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.



(A) **(B)**

Figure 4.44: Initial and final graph of the `assign_and_nvalues` constraint

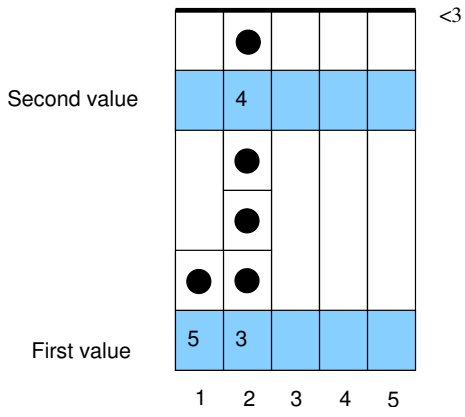


Figure 4.45: An assignment with at most two distinct values in parallel

- 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 atleast(N, VARIABLES, VALUE)

Argument(s)

N	:	int
VARIABLES	:	collection(var − dvar)
VALUE	:	int

Restriction(s) N ≥ 0
N $\leq$ |VARIABLES|
required(VARIABLES, var)

Purpose At least N variables of the VARIABLES collection are assigned to value VALUE.

Arc input(s) VARIABLES

Arc generator $SELF \mapsto$ collection(variables)

Arc arity 1

Arc constraint(s) variables.var = VALUE

Graph property(ies) **NARC** $\geq$ N

Example atleast$(2, \{\text{var} - 4, \text{var} - 2, \text{var} - 4, \text{var} - 5\}, 4)$

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.

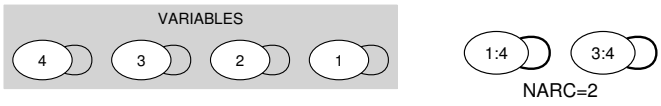


Figure 4.46: Initial and final graph of the atleast constraint

Graph model Since we use a unary arc constraint (VALUE is fixed) we employ the *SELF* 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 atleast 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: $\text{VAR}_i = \text{VALUE} \Leftrightarrow \text{S}_i$. The automaton counts the number of variables of the VARIABLES collection which are assigned to VALUE and finally checks that this number is greater than or equal to N.

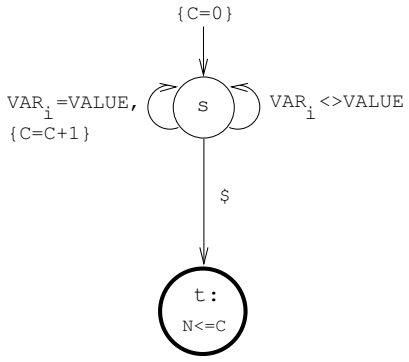


Figure 4.47: Automaton of the `atleast` constraint

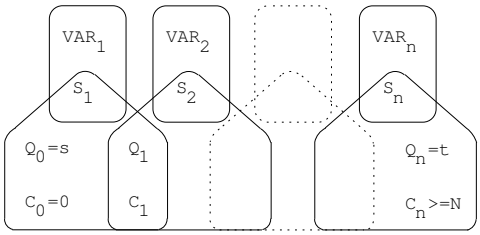


Figure 4.48: Hypergraph of the reformulation corresponding to the automaton of the `atleast` constraint

See also `atmost`, `among`, `exactly`.

Key words value constraint, at least, automaton, automaton with counters, alpha-acyclic constraint network(2).

4.24 atmost

Origin CHIP

Constraint atmost(N, VARIABLES, VALUE)

Argument(s)

N	:	int
VARIABLES	:	collection(var − dvar)
VALUE	:	int

Restriction(s) N ≥ 0
required(VARIABLES, var)

Purpose At most N variables of the VARIABLES collection are assigned to value VALUE.

Arc input(s) VARIABLES

Arc generator $SELF \mapsto$ collection(variables)

Arc arity 1

Arc constraint(s) variables.var = VALUE

Graph property(ies) **NARC** ≤ N

Example atmost(1, {var − 4, var − 2, var − 4, var − 5}, 2)

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.

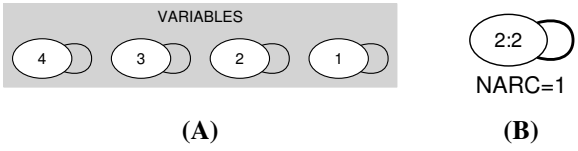


Figure 4.49: Initial and final graph of the atmost constraint

Graph model Since we use a unary arc constraint (VALUE is fixed) we employ the *SELF* 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 atmost constraint. To each variable $\mathtt{VAR}_i$ of the collection VARIABLES corresponds a 0-1 signature variable $\mathtt{S}_i$. The following signature constraint links $\mathtt{VAR}_i$ and $\mathtt{S}_i$: $\mathtt{VAR}_i = \mathtt{VALUE} \Leftrightarrow \mathtt{S}_i$. The automaton counts the number of variables of the VARIABLES collection which are assigned to VALUE and finally checks that this number is less than or equal to N.

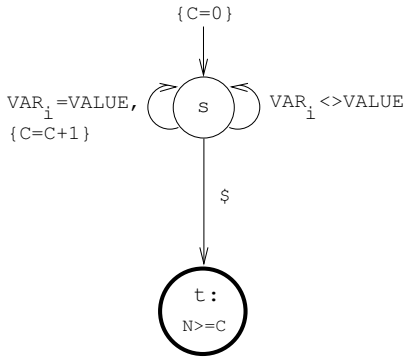


Figure 4.50: Automaton of the `atmost` constraint

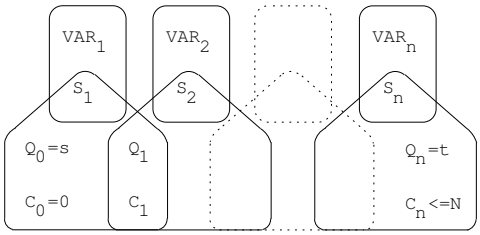


Figure 4.51: Hypergraph of the reformulation corresponding to the automaton of the `atmost` constraint

See also `atleast`, `among`, `exactly`, `cumulative`.

Key words value constraint, at most, automaton, automaton with counters, alpha-acyclic constraint network(2).

4.25 balance

Origin N. Beldiceanu

Constraint balance(BALANCE, VARIABLES)

Argument(s)

BALANCE	:	dvar
VARIABLES	:	collection(var − dvar)

Restriction(s)

BALANCE ≥ 0
BALANCE $\leq |$VARIABLES$|$
required(VARIABLES, var)

Purpose BALANCE 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 VARIABLES.

Arc input(s) VARIABLES

Arc generator $CLIQUE \mapsto$ collection(variables1, variables2)

Arc arity 2

Arc constraint(s) variables1.var = variables2.var

Graph property(ies) **RANGE_NSCC** = BALANCE

Example balance$\left(2, \left\{\begin{array}{l} \text{var} - 3, \\ \text{var} - 1, \\ \text{var} - 7, \\ \text{var} - 1, \\ \text{var} - 1 \end{array}\right\}\right)$

In this example, values 1, 3 and 7 are respectively used 3, 1 and 1 times. BALANCE 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 **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.

Automaton Figure 4.53 depicts the automaton associated to the balance 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 values, no matter how many distinct values will be used. In this case one will *push down* the maximum value of the first argument of the balance constraint.

See also balance_interval, balance_modulo, balance_partition, tree_range.

Key words value constraint, assignment, balanced assignment, automaton, automaton with array of counters, equivalence.

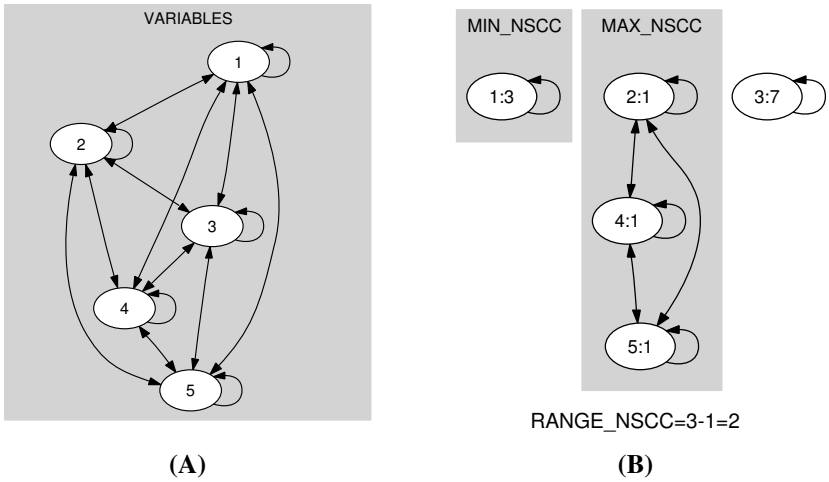


Figure 4.52: Initial and final graph of the `balance` constraint

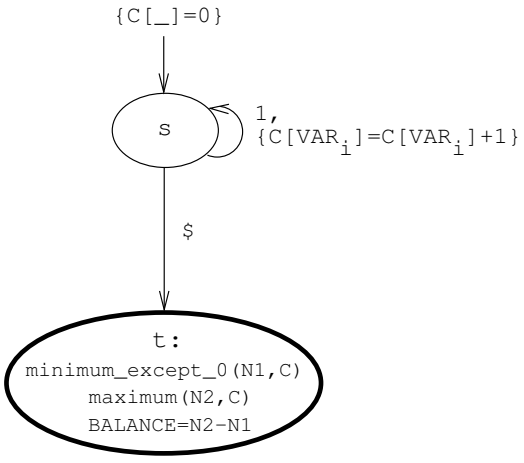


Figure 4.53: Automaton of the `balance` constraint

4.26 balance_interval

Origin Derived from balance.

Constraint balance_interval(BALANCE, VARIABLES, SIZE_INTERVAL)

Argument(s)

BALANCE	:	dvar
VARIABLES	:	collection(var − dvar)
SIZE_INTERVAL	:	int

Restriction(s)

BALANCE ≥ 0
BALANCE $\leq |$VARIABLES$|$
required(VARIABLES, var)
SIZE_INTERVAL > 0

Purpose Consider the largest set $\mathcal{S}_1$ (respectively the smallest set $\mathcal{S}_2$) of variables of the collection VARIABLES which take their value in a same interval $[$SIZE_INTERVAL $\cdot\, k$, SIZE_INTERVAL $\cdot\, k +$ SIZE_INTERVAL $- 1]$, where k is an integer. BALANCE is equal to the difference between the cardinality of $\mathcal{S}_2$ and the cardinality of $\mathcal{S}_1$.

Arc input(s) VARIABLES

Arc generator $CLIQUE \mapsto$ collection(variables1, variables2)

Arc arity 2

Arc constraint(s) variables1.var/SIZE_INTERVAL = variables2.var/SIZE_INTERVAL

Graph property(ies) **RANGE_NSCC** = BALANCE

Example

$$\text{balance\_interval}\left(\begin{array}{c} 3, \left\{\begin{array}{l} \text{var} - 6, \\ \text{var} - 4, \\ \text{var} - 3, \\ \text{var} - 3, \\ \text{var} - 4 \end{array}\right\}, 3 \end{array}\right)$$

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.

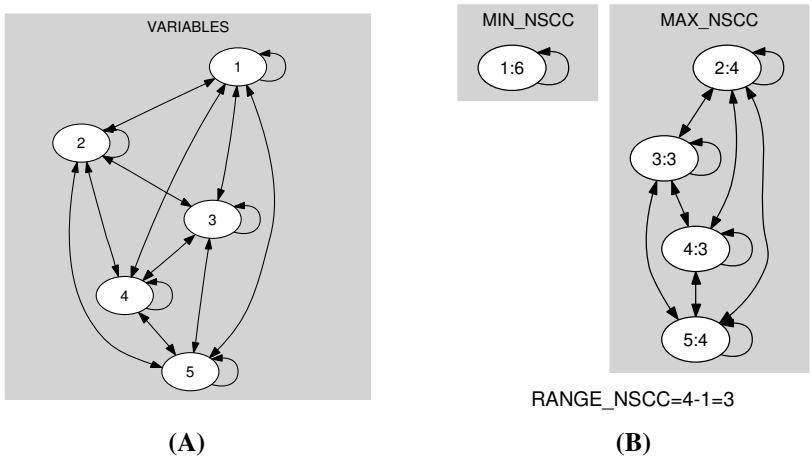


Figure 4.54: Initial and final graph of the `balance_interval` constraint

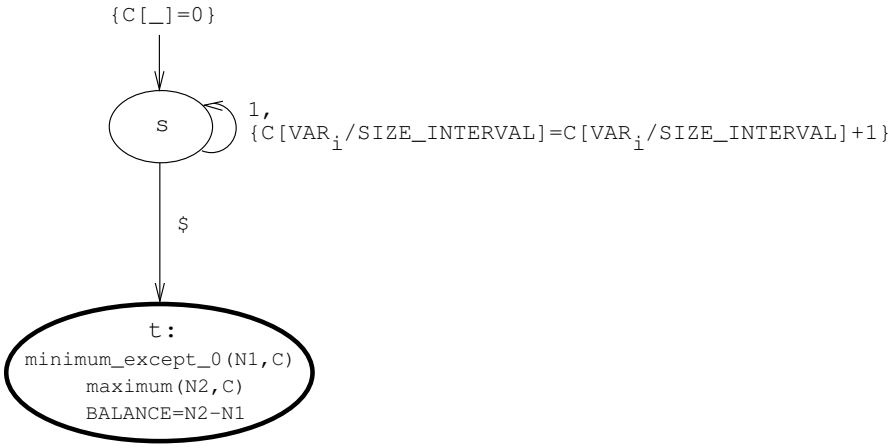


Figure 4.55: Automaton of the `balance_interval` constraint

Automaton Figure 4.55 depicts the automaton associated to the `balance_interval` constraint. To each item of the collection `VARIABLES` corresponds a signature variable $\mathtt{S}_i$, 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.

4.27 balance_modulo

Origin Derived from balance.

Constraint balance_modulo(BALANCE, VARIABLES, M)

Argument(s)

BALANCE	:	dvar
VARIABLES	:	collection(var − dvar)
M	:	int

Restriction(s)

BALANCE ≥ 0
BALANCE $\leq |$VARIABLES$|$
required(VARIABLES, var)
M > 0

Purpose Consider the largest set $\mathcal{S}_1$ (respectively the smallest set $\mathcal{S}_2$) of variables of the collection VARIABLES which have the same remainder when divided by M. BALANCE is equal to the difference between the cardinality of $\mathcal{S}_2$ and the cardinality of $\mathcal{S}_1$.

Arc input(s) VARIABLES

Arc generator $CLIQUE \mapsto$ collection(variables1, variables2)

Arc arity 2

Arc constraint(s) variables1.var mod M = variables2.var mod M

Graph property(ies) **RANGE_NSCC** = BALANCE

Example balance_modulo$\left(2, \left\{ \begin{array}{l} \text{var} - 6, \\ \text{var} - 1, \\ \text{var} - 7, \\ \text{var} - 1, \\ \text{var} - 5 \end{array} \right\}, 3 \right)$

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. BALANCE 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 **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.

Automaton Figure 4.57 depicts the automaton associated to the balance_modulo constraint. To each item of the collection VARIABLES corresponds a signature variable $\mathtt{S}_i$, which is equal to 1.

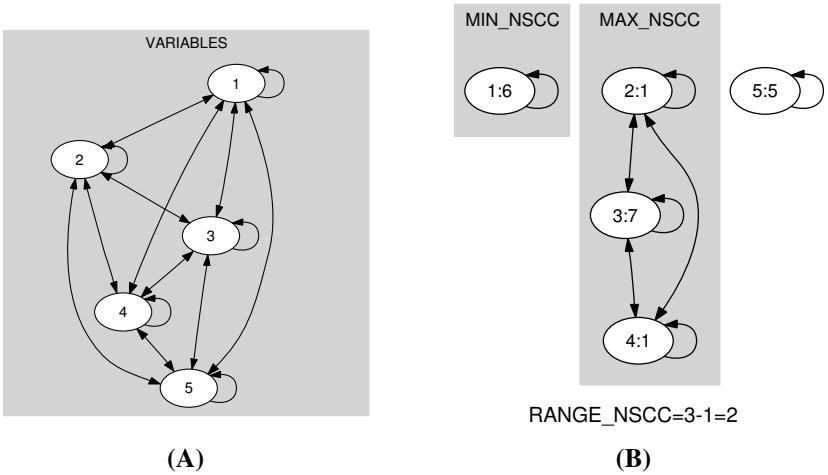


Figure 4.56: Initial and final graph of the balance_modulo constraint

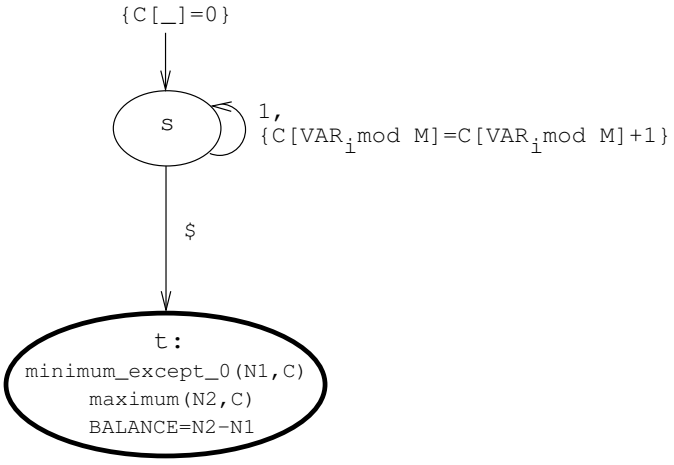


Figure 4.57: Automaton of the balance_modulo constraint

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.

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 $\leq |$VARIABLES$|$
required(VARIABLES, var)
required(PARTITIONS, p)
$|$PARTITIONS$| \geq 2$

Purpose Consider the largest set $\mathcal{S}_1$ (respectively the smallest set $\mathcal{S}_2$) 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 $\mathcal{S}_2$ and the cardinality of $\mathcal{S}_1$.

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

$$\texttt{balance\_partition}\left(\begin{array}{l} 1, \left\{\begin{array}{l}\texttt{var}-6,\\ \texttt{var}-2,\\ \texttt{var}-6,\\ \texttt{var}-4,\\ \texttt{var}-4\end{array}\right\},\\ \left\{\begin{array}{l}\texttt{p}-\{\texttt{val}-1,\texttt{val}-3\},\\ \texttt{p}-\{\texttt{val}-4\},\\ \texttt{p}-\{\texttt{val}-2,\texttt{val}-6\}\end{array}\right\}\end{array}\right)$$

In this example values $6, 2, 6, 4, 4$ are respectively associated to the partitions $\texttt{p} - \{\texttt{val} - 2, \texttt{val} - 6\}$ and $\texttt{p} - \{\texttt{val} - 4\}$. Partitions $\texttt{p} - \{\texttt{val} - 4\}$ and $\texttt{p} - \{\texttt{val} - 2, \texttt{val} - 6\}$ 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 *balanced assignment* of values, no matter how many distinct partitions will be used. In this case one will *push down* the maximum value of the first argument of the `balance_partition` constraint.

See also `balance`.

Key words value constraint, partition, assignment, balanced assignment, equivalence.

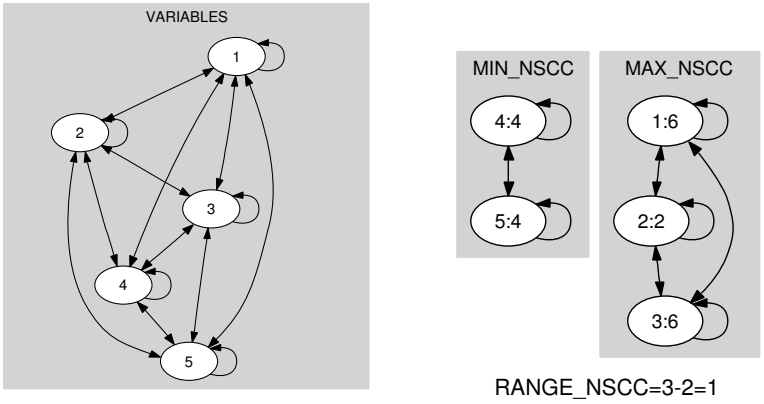


Figure 4.58: Initial and final graph of the `balance_partition` constraint

4.29 bin_packing

Origin Derived from cumulative.

Constraint bin_packing(CAPACITY, ITEMS)

Argument(s)

CAPACITY	:	int
ITEMS	:	collection(bin − dvar, weight − int)

Restriction(s)

CAPACITY ≥ 0
required(ITEMS, [bin, weight])
ITEMS.weight ≥ 0
ITEMS.weight $\leq$ CAPACITY

Purpose Given several items of the collection ITEMS (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 CAPACITY.

Arc input(s) ITEMS ITEMS

Arc generator $PRODUCT \mapsto$ collection(items1, items2)

Arc arity 2

Arc constraint(s) items1.bin = items2.bin

Sets

$$\text{SUCC} \mapsto \left[\begin{array}{l} \texttt{source}, \\ \texttt{variables} - \texttt{col}\left(\begin{array}{l} \texttt{VARIABLES} - \texttt{collection}(\texttt{var} - \texttt{dvar}), \\ [\texttt{item}(\texttt{var} - \texttt{ITEMS.weight})] \end{array} \right) \end{array} \right]$$

Constraint(s) on sets sum_ctr(variables, $\leq$, CAPACITY)

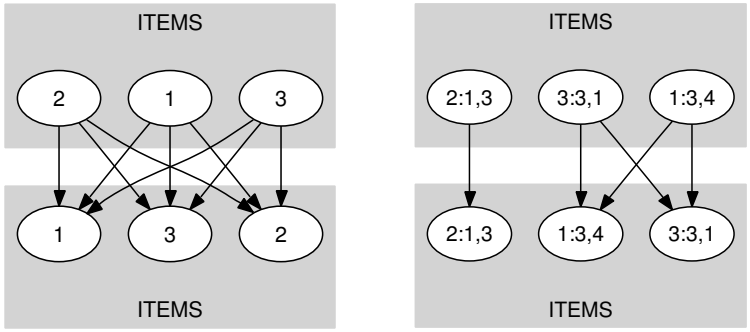
Example

$$\texttt{bin\_packing}\left(5, \left\{ \begin{array}{ll} \texttt{bin} - 3 & \texttt{weight} - 4, \\ \texttt{bin} - 1 & \texttt{weight} - 3, \\ \texttt{bin} - 3 & \texttt{weight} - 1 \end{array} \right\} \right)$$

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 bin_packing 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 CAPACITY 5. Figure 4.60 shows the solution associated to the previous example.

Graph model We enforce the sum_ctr constraint on the weight of the items that are assigned to the same bin.

Automaton Figure 4.61 depicts the automaton associated to the bin_packing constraint. To each item of the collection ITEMS corresponds a signature variable S_i, which is equal to 1.



(A) **(B)**

Figure 4.59: Initial and final graph of the `bin_packing` constraint

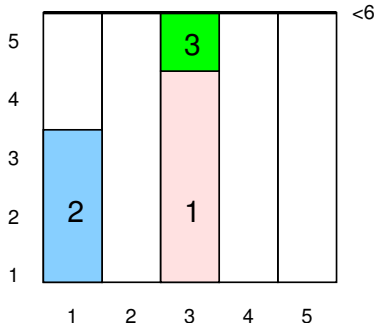


Figure 4.60: Bin-packing solution

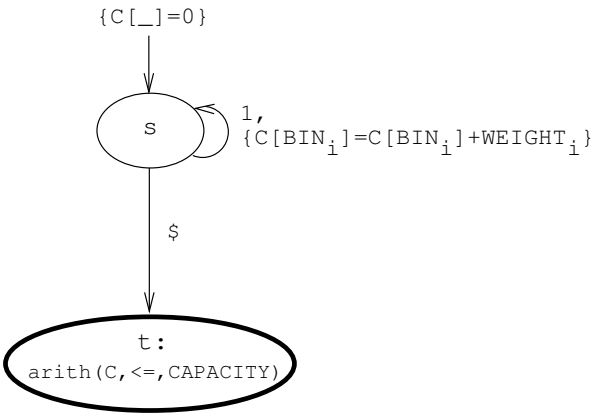


Figure 4.61: Automaton of the `bin_packing` constraint

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 weigths 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.

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 ≥ 0
required(NODES, [index, succ])
NODES.index ≥ 1
NODES.index $\leq |$NODES$|$
distinct(NODES, index)
NODES.succ ≥ 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)

- $\mathbf{MAX\_NSCC} \leq 1$
- $\mathbf{NCC} =$ NTREES
- $\mathbf{MAX\_ID} \leq 2$

Example

$$\texttt{binary\_tree}\left(\begin{array}{l} 2, \left\{\begin{array}{ll} \texttt{index} - 1 & \texttt{succ} - 1, \\ \texttt{index} - 2 & \texttt{succ} - 3, \\ \texttt{index} - 3 & \texttt{succ} - 5, \\ \texttt{index} - 4 & \texttt{succ} - 7, \\ \texttt{index} - 5 & \texttt{succ} - 1, \\ \texttt{index} - 6 & \texttt{succ} - 1, \\ \texttt{index} - 7 & \texttt{succ} - 7, \\ \texttt{index} - 8 & \texttt{succ} - 5 \end{array}\right\} \end{array}\right)$$

Parts (A) and (B) of Figure 4.62 respectively show the initial and final graph. Since we use the $\mathbf{NCC}$ graph property, we display the two connected components of the final graph. Each of them corresponds to a binary tree. Since we use the $\mathbf{MAX\_ID}$ 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 NTREES $= \mathbf{NCC} = 2$ and since $\mathbf{MAX\_ID} = 2$.

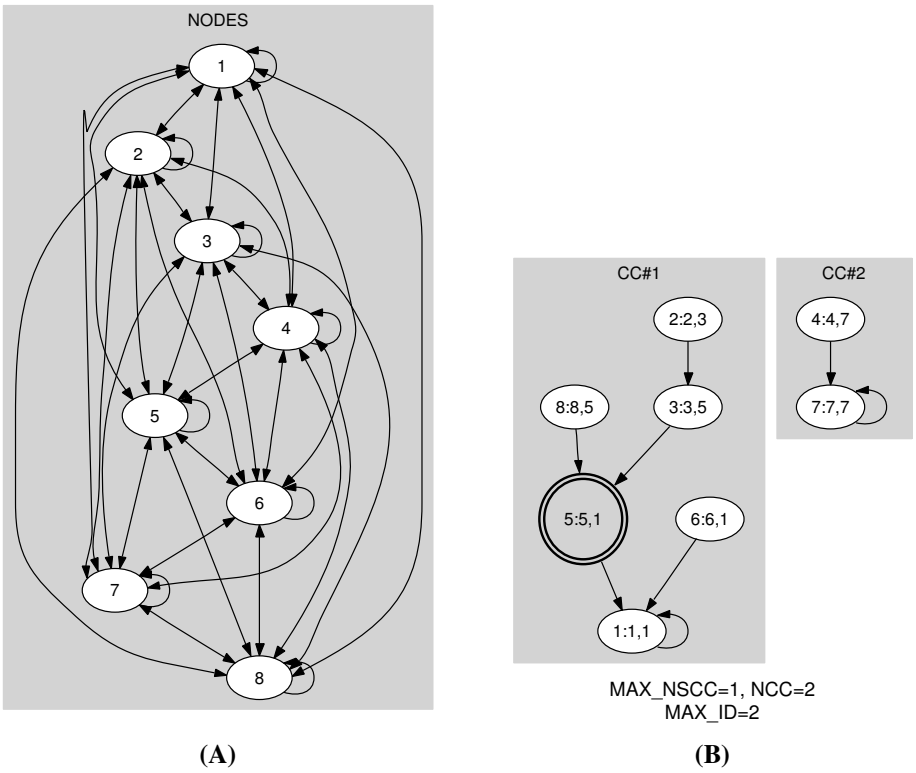


Figure 4.62: Initial and final graph of the `binary_tree` constraint

Graph model We use the same graph constraint as for the `tree` constraint, except that we add the graph property $\mathbf{MAX\_ID} \leq 2$ which constraints the maximum in-degree of the final graph to not exceed 2. $\mathbf{MAX\_ID}$ 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.

4.31 cardinality_atleast

Origin Derived from global_cardinality.

Constraint cardinality_atleast(ATLEAST, VARIABLES, VALUES)

Argument(s)

ATLEAST	:	dvar
VARIABLES	:	collection(var − dvar)
VALUES	:	collection(val − int)

Restriction(s)
ATLEAST ≥ 0
ATLEAST $\leq |$VARIABLES$|$
required(VARIABLES, var)
required(VALUES, val)
distinct(VALUES, val)

Purpose ATLEAST is the minimum number of time that a value of VALUES is taken by the variables of the collection VARIABLES.

Arc input(s) VARIABLES VALUES

Arc generator $PRODUCT \mapsto$ collection(variables, values)

Arc arity 2

Arc constraint(s) variables.var $\neq$ values.val

Graph property(ies) **MAX_ID** $=$ |VARIABLES| $-$ ATLEAST

Example cardinality_atleast $\left(\begin{array}{l} 1, \{\text{var} - 3, \text{var} - 3, \text{var} - 8\}, \\ \{\text{val} - 3, \text{val} - 8\} \end{array} \right)$

In this example, values 3 and 8 are respectively used 2, and 1 times. Therefore ATLEAST 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 **MAX_ID** graph property, the vertex with the maximum number of predecessor is stressed with a double circle.

Graph model Using directly the graph property **MIN_ID** $=$ ATLEAST 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 cardinality_atleast 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$_i$.

Usage An application of this constraint is to enforce a minimum use of values.

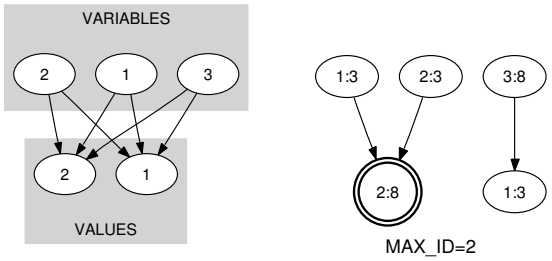


Figure 4.63: Initial and final graph of the `cardinality_atleast` constraint

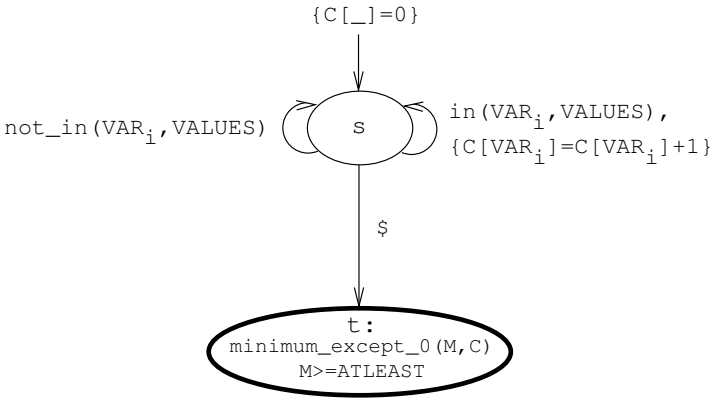


Figure 4.64: Automaton of the `cardinality_atleast` constraint

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.

4.32 cardinality_atmost

Origin Derived from global_cardinality.

Constraint cardinality_atmost(ATMOST, VARIABLES, VALUES)

Argument(s)

ATMOST	:	dvar
VARIABLES	:	collection(var − dvar)
VALUES	:	collection(val − int)

Restriction(s)
ATMOST ≥ 0
ATMOST $\leq |$VARIABLES$|$
required(VARIABLES, var)
required(VALUES, val)
distinct(VALUES, val)

Purpose ATMOST is the maximum number of occurrences of each value of VALUES within the variables of the collection VARIABLES.

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) **MAX_ID** = ATMOST

Example

$$\texttt{cardinality\_atmost}\left(\begin{array}{l} 2, \left\{\begin{array}{l}\texttt{var}-2,\\ \texttt{var}-1,\\ \texttt{var}-7,\\ \texttt{var}-1,\\ \texttt{var}-2\end{array}\right\},\\ \left\{\begin{array}{l}\texttt{val}-5,\\ \texttt{val}-7,\\ \texttt{val}-2,\\ \texttt{val}-9\end{array}\right\}\end{array}\right)$$

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 $\texttt{VAR}_i$ of the collection VARIABLES corresponds a 0-1 signature variable $\texttt{S}_i$. The following signature constraint links $\texttt{VAR}_i$ and $\texttt{S}_i$: $\texttt{VAR}_i \in \texttt{VALUES} \Leftrightarrow \texttt{S}_i$.

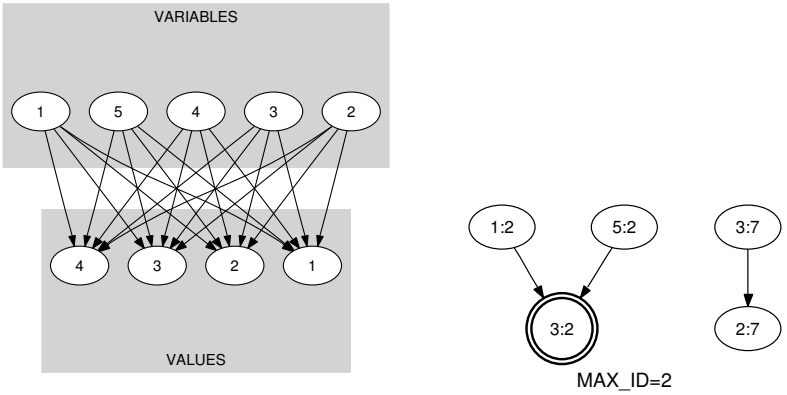


Figure 4.65: Initial and final graph of the `cardinality_atmost` constraint

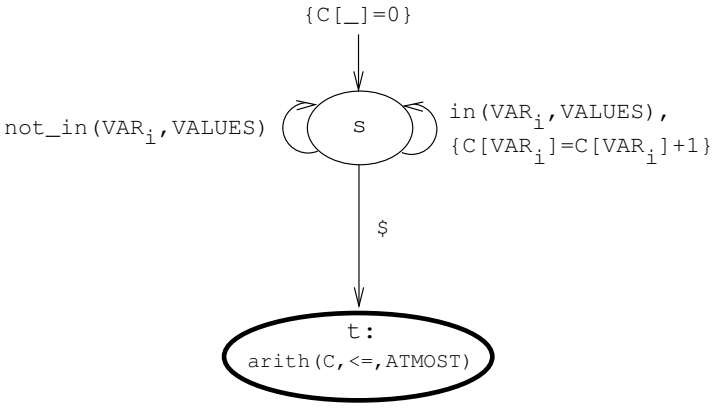


Figure 4.66: Automaton of the `cardinality_atmost` constraint

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 `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 most, automaton, automaton with array of counters, acyclic, bipartite, no_loop.

4.33 cardinality_atmost_partition

Origin Derived from global_cardinality.

Constraint cardinality_atmost_partition(ATMOST, VARIABLES, PARTITIONS)

Type(s) VALUES : collection(val − int)

Argument(s)

ATMOST	:	dvar
VARIABLES	:	collection(var − dvar)
PARTITIONS	:	collection(p − VALUES)

Restriction(s)
required(VALUES, val)
distinct(VALUES, val)
ATMOST ≥ 0
ATMOST $\leq |$VARIABLES$|$
required(VARIABLES, var)
required(PARTITIONS, p)
$|$PARTITIONS$| \geq 2$

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 $PRODUCT \mapsto$ collection(variables, partitions)

Arc arity 2

Arc constraint(s) in(variables.var, partitions.p)

Graph property(ies) MAX_ID = ATMOST

Example

$$\texttt{cardinality\_atmost\_partition}\left(\begin{array}{l} 2, \left\{\begin{array}{l} \texttt{var}-2,\\ \texttt{var}-3,\\ \texttt{var}-7,\\ \texttt{var}-1,\\ \texttt{var}-6,\\ \texttt{var}-0 \end{array}\right\},\\ \left\{\begin{array}{l} \texttt{p}-\{\texttt{val}-1, \texttt{val}-3\},\\ \texttt{p}-\{\texttt{val}-4\},\\ \texttt{p}-\{\texttt{val}-2, \texttt{val}-6\} \end{array}\right\} \end{array}\right)$$

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.

See also `global_cardinality`, `in`.

Key words value constraint, partition, at most, acyclic, bipartite, no_loop.

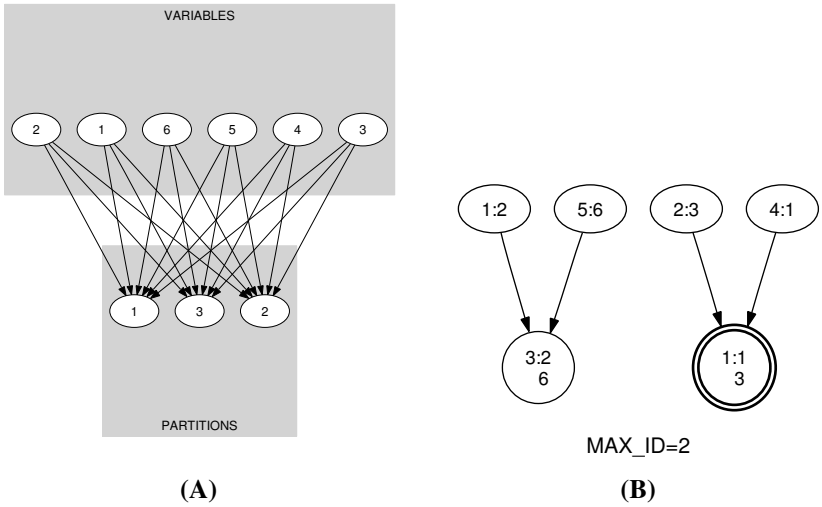


Figure 4.67: Initial and final graph of the `cardinality_atmost_partition` constraint

4.34 change

Origin CHIP

Constraint change(NCHANGE, VARIABLES, CTR)

Synonym(s) nbchanges, similarity.

Argument(s)

NCHANGE	:	dvar
VARIABLES	:	collection(var − dvar)
CTR	:	atom

Restriction(s)

NCHANGE ≥ 0
NCHANGE $<$ |VARIABLES|
required(VARIABLES, var)
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 $PATH \mapsto$ collection(variables1, variables2)

Arc arity 2

Arc constraint(s) variables1.var CTR variables2.var

Graph property(ies) **NARC** = NCHANGE

Example

$$\texttt{change}\left(3, \left\{\begin{array}{l}\texttt{var}-4,\\ \texttt{var}-4,\\ \texttt{var}-3,\\ \texttt{var}-4,\\ \texttt{var}-1\end{array}\right\}, \neq\right)$$

$$\texttt{change}\left(1, \left\{\begin{array}{l}\texttt{var}-1,\\ \texttt{var}-2,\\ \texttt{var}-4,\\ \texttt{var}-3,\\ \texttt{var}-7\end{array}\right\}, >\right)$$

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 change constraint. To each pair of consecutive variables $(\texttt{VAR}_i, \texttt{VAR}_{i+1})$ of the collection VARIABLES corresponds a 0-1 signature variable $\texttt{S}_i$. The following signature constraint links $\texttt{VAR}_i$, $\texttt{VAR}_{i+1}$ and $\texttt{S}_i$: $\texttt{VAR}_i$ CTR $\texttt{VAR}_{i+1} \Leftrightarrow \texttt{S}_i$.

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 similarity constraint. The difference consists of replacing the arithmetic constraint CTR by a binary constraint. When CTR is equal to $\neq$ this constraint is called nbchanges in [40].

Algorithm [65].

Used in pattern.

See also smooth, change_partition, change_pair, circular_change, longest_change.

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.

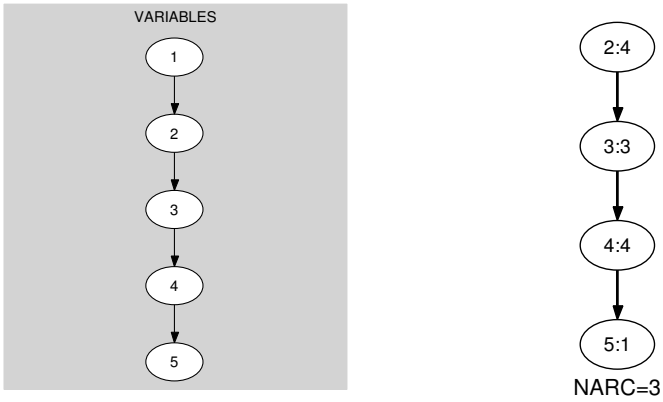


Figure 4.68: Initial and final graph of the change constraint

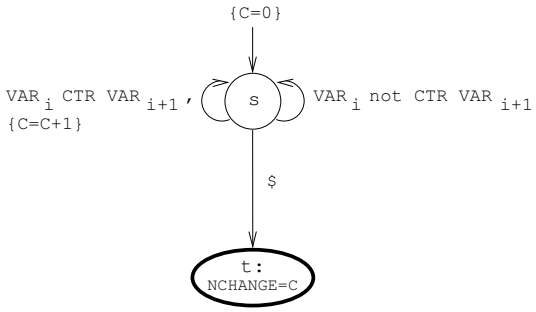


Figure 4.69: Automaton of the change constraint

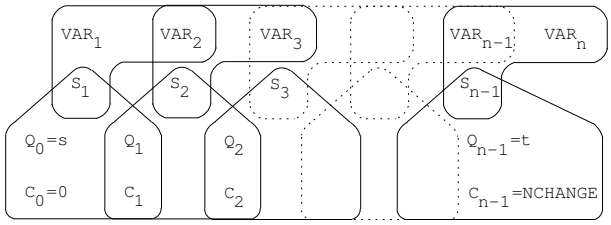


Figure 4.70: Hypergraph of the reformulation corresponding to the automaton of the change constraint

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 ≥ 0
NB_PERIOD_CONTINUITY ≥ 0
MIN_SIZE_CHANGE ≥ 0
MAX_SIZE_CHANGE $\geq$ MIN_SIZE_CHANGE
MIN_SIZE_CONTINUITY ≥ 0
MAX_SIZE_CONTINUITY $\geq$ MIN_SIZE_CONTINUITY
NB_CHANGE ≥ 0
NB_CONTINUITY ≥ 0
required(VARIABLES, var)
CTR $\in [=, \neq, <, \geq, >, \leq]$

Purpose

On the one hand a *change* is defined by the fact that constraint VARIABLES$[i]$.var CTR VARIABLES$[i+1]$.var holds.
On the other hand a *continuity* is defined by the fact that constraint VARIABLES$[i]$.var CTR VARIABLES$[i+1]$.var does not hold.
A *period of change* on variables

$$\texttt{VARIABLES}[i].\texttt{var}, \texttt{VARIABLES}[i+1].\texttt{var}, \ldots, \texttt{VARIABLES}[j].\texttt{var} \ (i < j)$$

is defined by the fact that all constraints VARIABLES$[k]$.var CTR VARIABLES$[k+1]$.var hold for $k \in [i, j-1]$.
A *period of continuity* on variables

$$\texttt{VARIABLES}[i].\texttt{var}, \texttt{VARIABLES}[i+1].\texttt{var}, \ldots, \texttt{VARIABLES}[j].\texttt{var} \ (i < j)$$

is defined by the fact that all constraints VARIABLES$[k]$.var CTR VARIABLES$[k+1]$.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.

Arc input(s) VARIABLES

Arc generator $PATH \mapsto \texttt{collection}(\texttt{variables1}, \texttt{variables2})$

Arc arity 2

Arc constraint(s) variables1.var CTR variables2.var

Graph property(ies)
- $\mathbf{NCC} = \texttt{NB\_PERIOD\_CHANGE}$
- $\mathbf{MIN\_NCC} = \texttt{MIN\_SIZE\_CHANGE}$
- $\mathbf{MAX\_NCC} = \texttt{MAX\_SIZE\_CHANGE}$
- $\mathbf{NARC} = \texttt{NB\_CHANGE}$

Arc input(s) VARIABLES

Arc generator $PATH \mapsto \texttt{collection}(\texttt{variables1}, \texttt{variables2})$

Arc arity 2

Arc constraint(s) variables1.var$\neg$ CTR variables2.var

Graph property(ies)

- $\mathbf{NCC} =$ NB_PERIOD_CONTINUITY
- $\mathbf{MIN\_NCC} =$ MIN_SIZE_CONTINUITY
- $\mathbf{MAX\_NCC} =$ MAX_SIZE_CONTINUITY
- $\mathbf{NARC} =$ NB_CONTINUITY

Example

$$\texttt{change\_continuity}\left(\begin{array}{l} 3,2,2,4,2,4,6,4, \left\{\begin{array}{l}\texttt{var}-1,\\ \texttt{var}-3,\\ \texttt{var}-1,\\ \texttt{var}-8,\\ \texttt{var}-8,\\ \texttt{var}-4,\\ \texttt{var}-7,\\ \texttt{var}-7,\\ \texttt{var}-7,\\ \texttt{var}-7,\\ \texttt{var}-2\end{array}\right\}, \neq \end{array}\right)$$

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 $\mathtt{S}_i$ for each pair of consecutive variables $(\mathtt{VAR}_i, \mathtt{VAR}_{i+1})$ of the collection VARIABLES. The following signature constraint links $\mathtt{VAR}_i$, $\mathtt{VAR}_{i+1}$ and $\mathtt{S}_i$: $\mathtt{VAR}_i$ CTR $\mathtt{VAR}_{i+1} \Leftrightarrow \mathtt{S}_i$.

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.

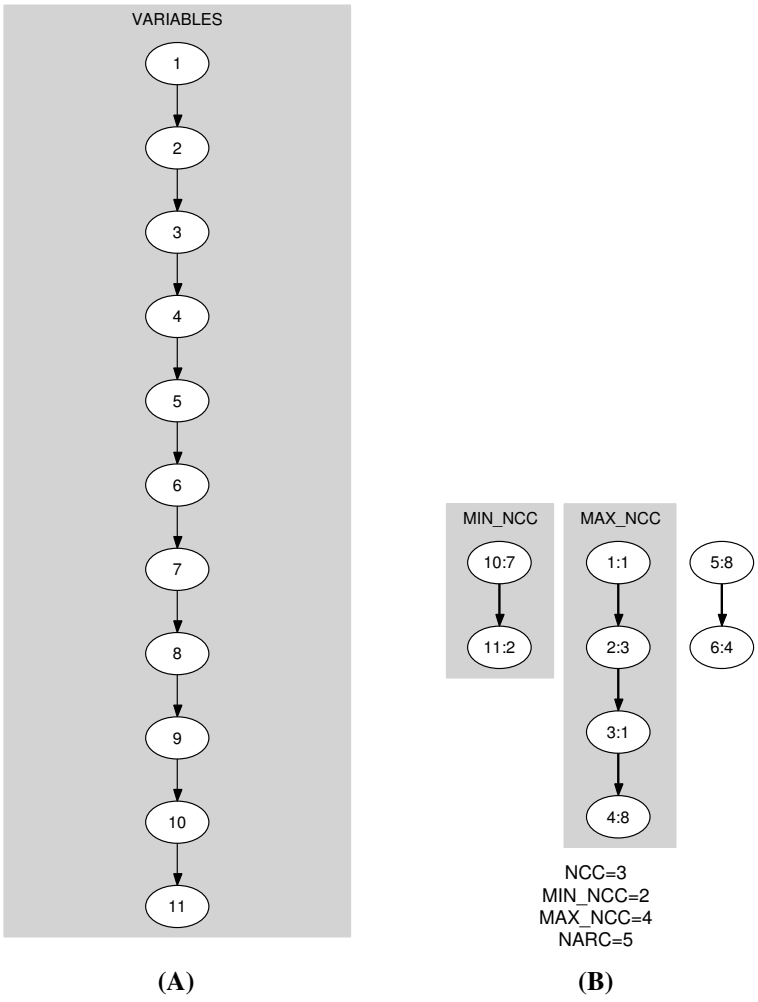


Figure 4.72: Initial and final graph of the `change_continuity` constraint

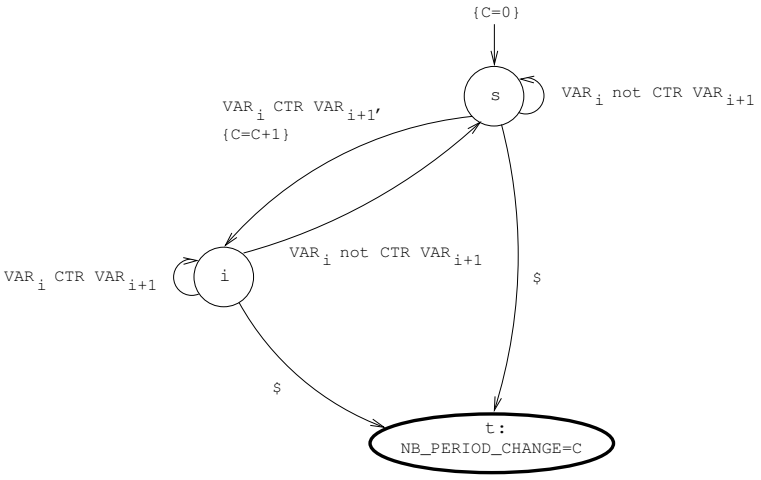


Figure 4.73: Automaton for the NB_PERIOD_CHANGE parameter of the change_continuity constraint

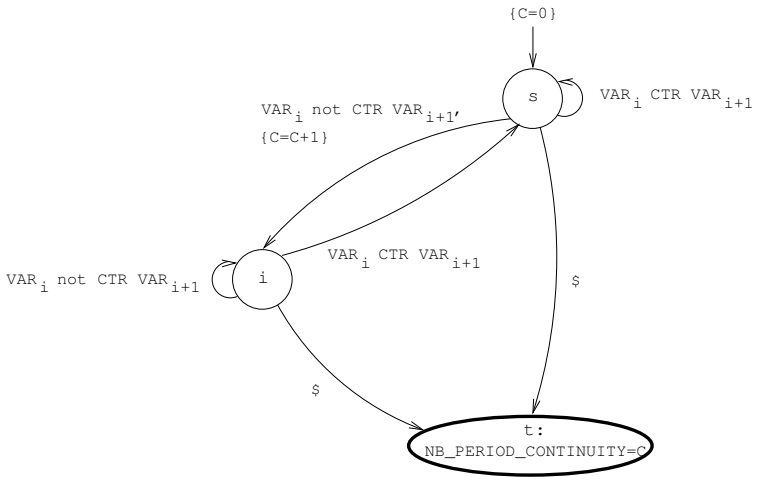


Figure 4.74: Automaton for the NB_PERIOD_CONTINUITY parameter of the change_continuity constraint

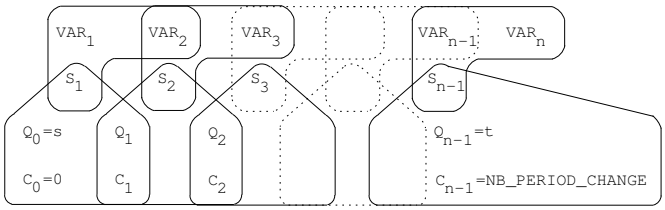


Figure 4.75: Hypergraph of the reformulation corresponding to the automaton of the NB_PERIOD_CHANGE parameter of the change_continuity constraint

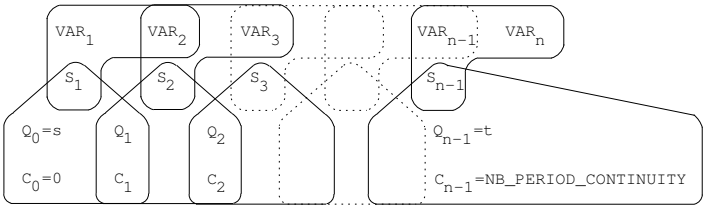


Figure 4.76: Hypergraph of the reformulation corresponding to the automaton of the NB_PERIOD_CONTINUITY parameter of the change_continuity constraint

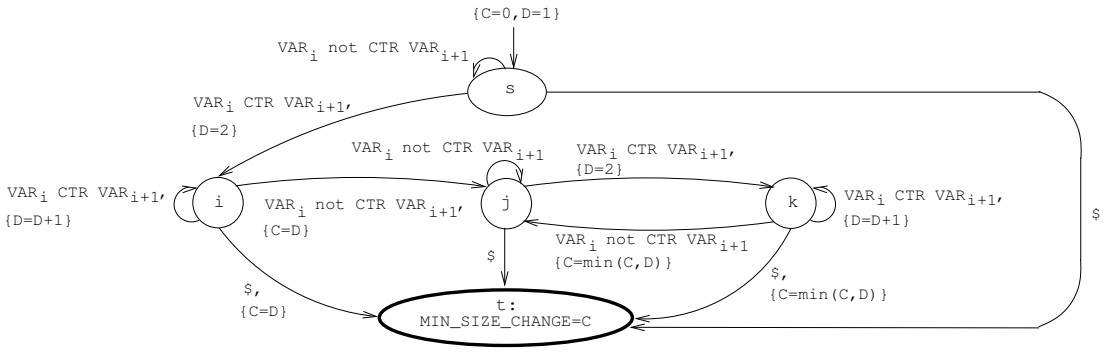


Figure 4.77: Automaton for the MIN_SIZE_CHANGE parameter of the change_continuity constraint

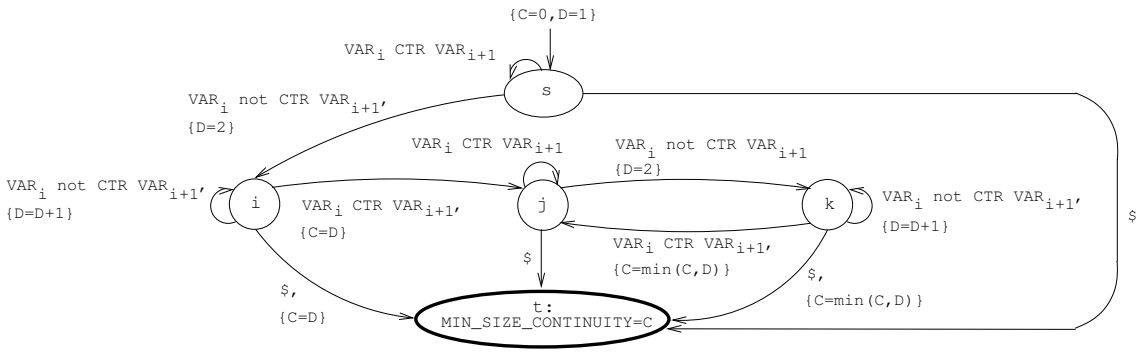


Figure 4.78: Automaton for the MIN_SIZE_CONTINUITY parameter of the change_continuity constraint

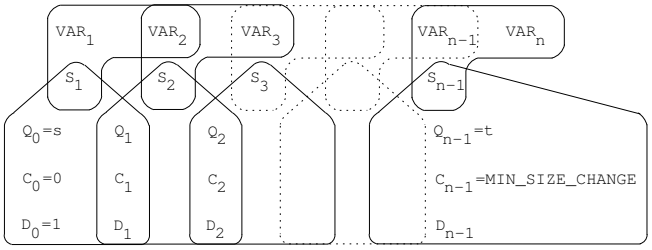


Figure 4.79: Hypergraph of the reformulation corresponding to the automaton of the MIN_SIZE_CHANGE parameter of the change_continuity constraint

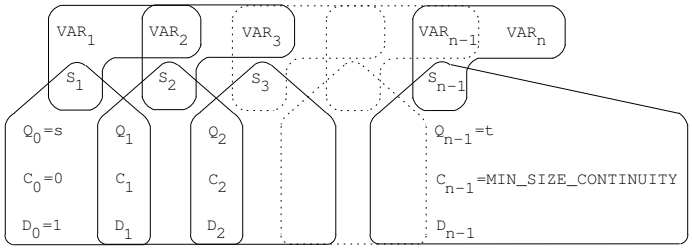


Figure 4.80: Hypergraph of the reformulation corresponding to the automaton of the MIN_SIZE_CONTINUITY parameter of the change_continuity constraint

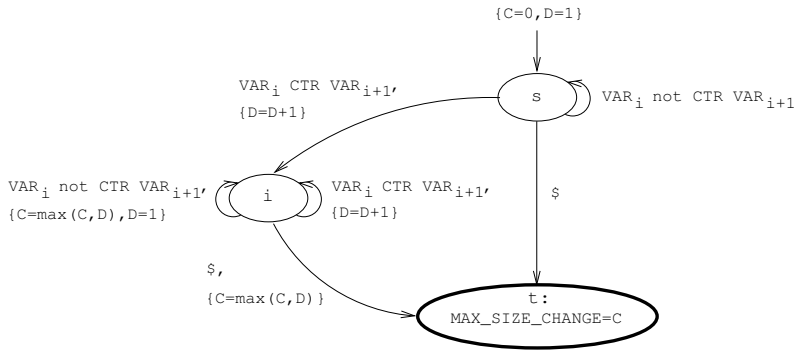


Figure 4.81: Automaton for the MAX_SIZE_CHANGE parameter of the change_continuity constraint

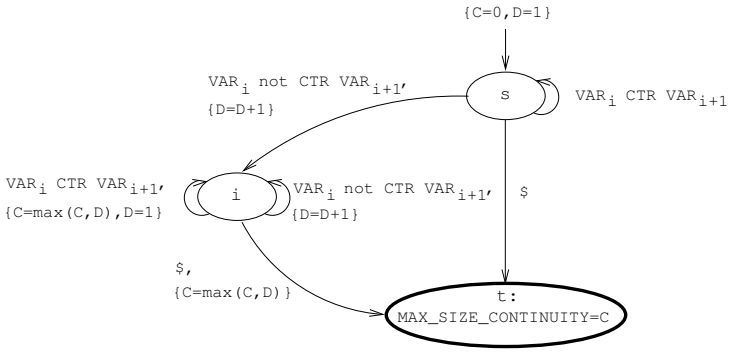


Figure 4.82: Automaton for the MAX_SIZE_CONTINUITY parameter of the change_continuity constraint

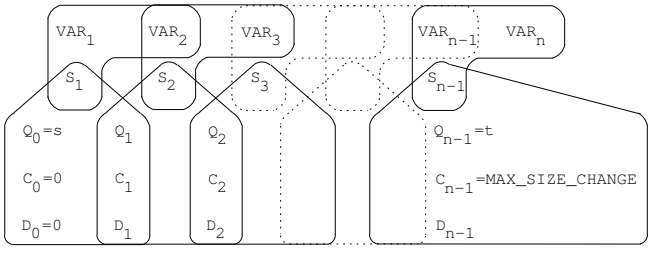


Figure 4.83: Hypergraph of the reformulation corresponding to the automaton of the MAX_SIZE_CHANGE parameter of the change_continuity constraint

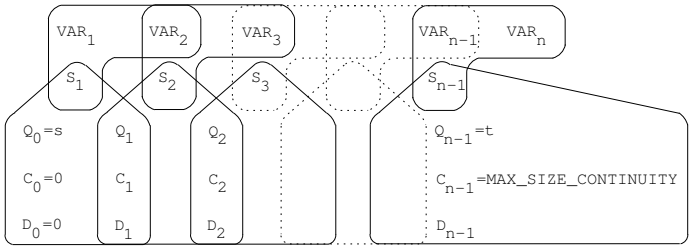


Figure 4.84: Hypergraph of the reformulation corresponding to the automaton of the MAX_SIZE_CONTINUITY parameter of the change_continuity constraint

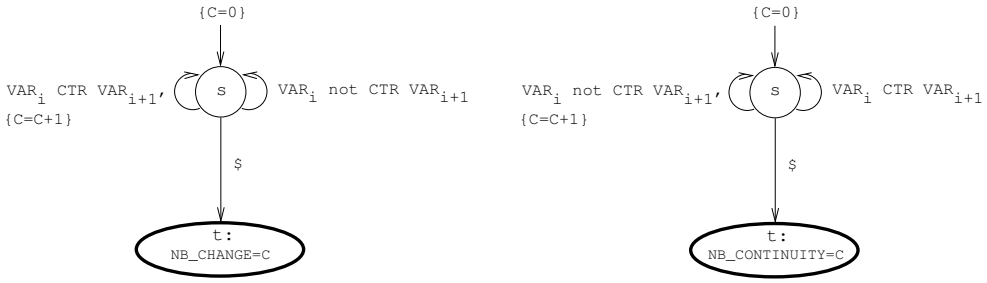


Figure 4.85: Automata for the NB_CHANGE and NB_CONTINUITY parameters of the change_continuity constraint

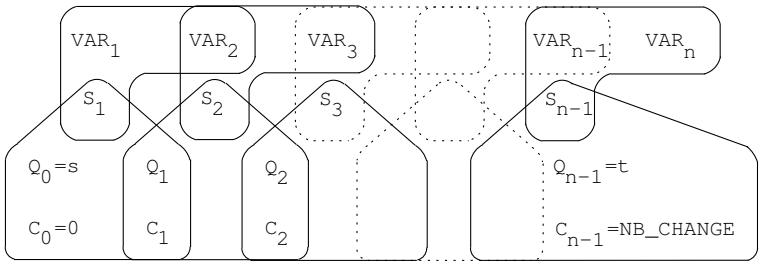


Figure 4.86: Hypergraph of the reformulation corresponding to the automaton of the NB_CHANGE parameter of the change_continuity constraint

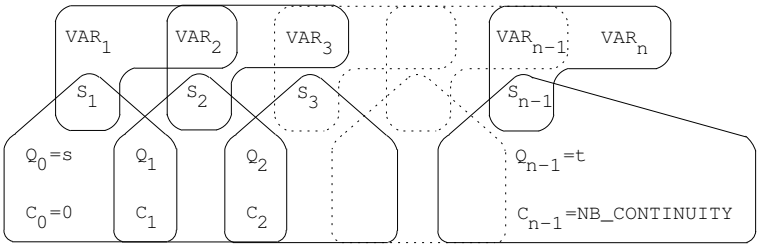


Figure 4.87: Hypergraph of the reformulation corresponding to the automaton of the NB_CONTINUITY parameter of the change_continuity constraint

4.36 change_pair

Origin Derived from change.

Constraint change_pair(NCHANGE, PAIRS, CTRX, CTRY)

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: $(X_1$ CTRX $X_2) \vee (Y_1$ 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 $PATH \mapsto$ collection(pairs1, pairs2)

Arc arity 2

Arc constraint(s) pairs1.x CTRX pairs2.x $\vee$ pairs1.y CTRY pairs2.y

Graph property(ies) NARC = NCHANGE

Example

$$\texttt{change\_pair}\left(\begin{array}{l} 3, \left\{\begin{array}{ll} \texttt{x}-3 & \texttt{y}-5, \\ \texttt{x}-3 & \texttt{y}-7, \\ \texttt{x}-3 & \texttt{y}-7, \\ \texttt{x}-3 & \texttt{y}-8, \\ \texttt{x}-3 & \texttt{y}-4, \\ \texttt{x}-3 & \texttt{y}-7, \\ \texttt{x}-1 & \texttt{y}-3, \\ \texttt{x}-1 & \texttt{y}-6, \\ \texttt{x}-1 & \texttt{y}-6, \\ \texttt{x}-3 & \texttt{y}-7 \end{array}\right\}, \neq, > \end{array}\right)$$

In the previous example we have the following 3 changes:

- One change between pairs x − 3 y − 8 and x − 3 y − 4,
- One change between pairs x − 3 y − 7 and x − 1 y − 3,
- One change between pairs x − 1 y − 6 and x − 3 y − 7.

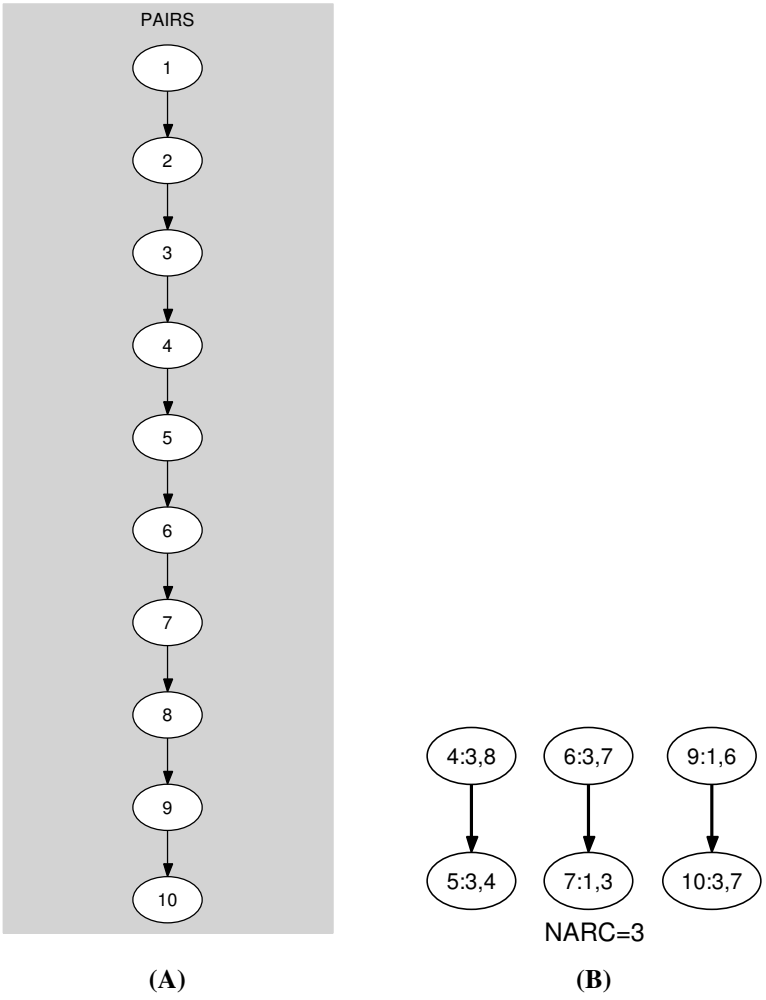


Figure 4.88: Initial and final graph of the `change_pair` constraint

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 $((\mathtt{X}_i, \mathtt{Y}_i), (\mathtt{X}_{i+1}, \mathtt{Y}_{i+1}))$ of the collection `PAIRS` corresponds a 0-1 signature variable $\mathtt{S}_i$. The following signature constraint links $\mathtt{X}_i$, $\mathtt{Y}_i$, $\mathtt{X}_{i+1}$, $\mathtt{Y}_{i+1}$ and $\mathtt{S}_i$: $(\mathtt{X}_i \ \mathtt{CTRX} \ \mathtt{X}_{i+1}) \vee (\mathtt{Y}_i \ \mathtt{CTRY} \ \mathtt{Y}_{i+1}) \Leftrightarrow \mathtt{S}_i$.

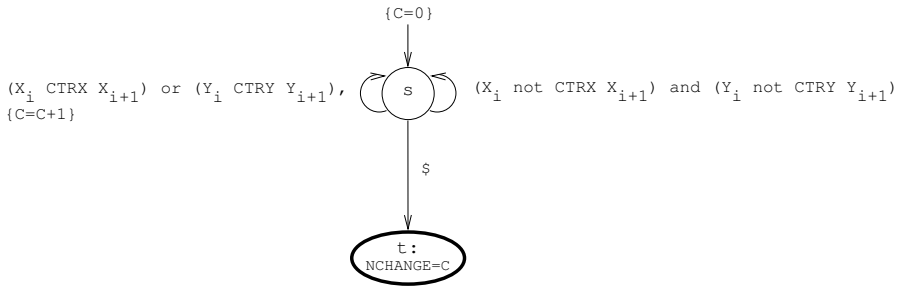


Figure 4.89: Automaton of the `change_pair` constraint

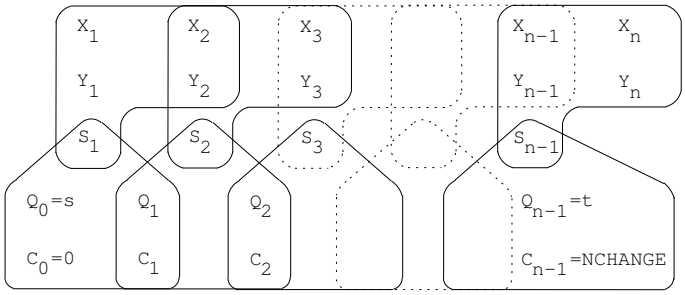


Figure 4.90: Hypergraph of the reformulation corresponding to the automaton of the `change_pair` constraint

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.

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 $PATH \mapsto$ collection(variables1, variables2)

Arc arity 2

Arc constraint(s) in_same_partition(variables1.var, variables2.var, PARTITIONS)

Graph property(ies) $\mathbf{NARC} =$ NCHANGE

Example

$$\texttt{change\_partition}\left(\begin{array}{l} 2, \left\{\begin{array}{l}\texttt{var}-6,\\ \texttt{var}-6,\\ \texttt{var}-2,\\ \texttt{var}-1,\\ \texttt{var}-3,\\ \texttt{var}-3,\\ \texttt{var}-1,\\ \texttt{var}-6,\\ \texttt{var}-2,\\ \texttt{var}-2,\\ \texttt{var}-2\end{array}\right\},\\ \left\{\begin{array}{l}\texttt{p}-\{\texttt{val}-1, \texttt{val}-3\},\\ \texttt{p}-\{\texttt{val}-4\},\\ \texttt{p}-\{\texttt{val}-2, \texttt{val}-6\}\end{array}\right\}\end{array}\right)$$

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.

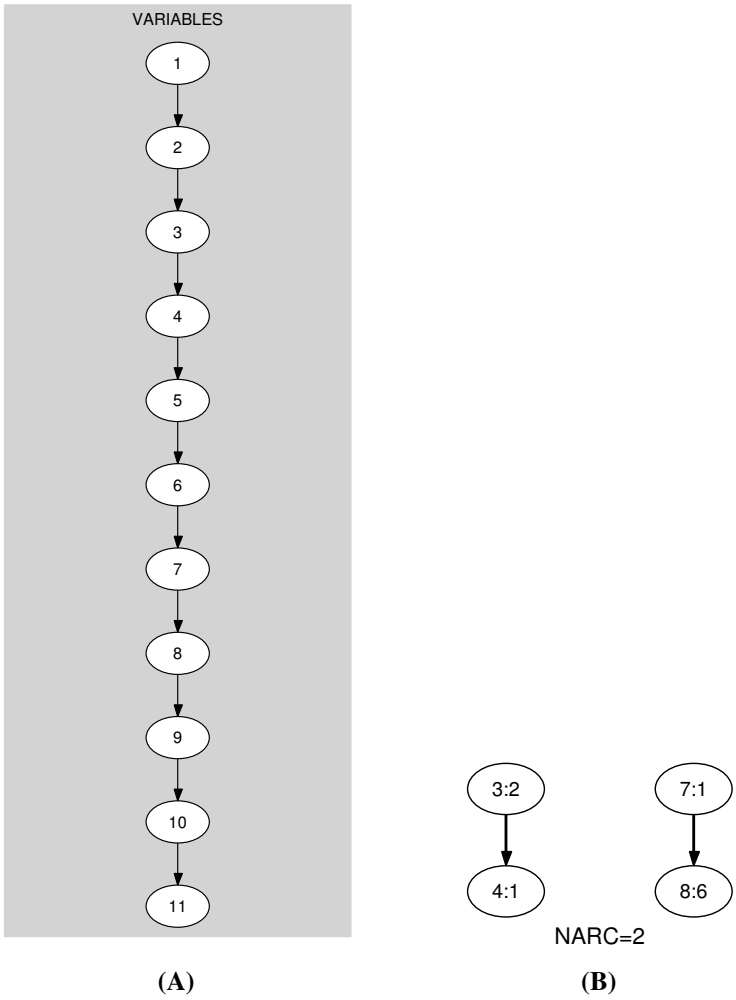


Figure 4.91: Initial and final graph of the `change_partition` constraint

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.

4.38 circuit

Origin [2]

Constraint circuit(NODES)

Synonym(s) atour, cycle.

Argument(s)

NODES	:	collection(index − int, succ − dvar)

Restriction(s)
required(NODES, [index, succ])
NODES.index ≥ 1
NODES.index ≤ |NODES|
distinct(NODES, index)
NODES.succ ≥ 1
NODES.succ ≤ |NODES|

Purpose Enforce to cover a digraph G described by the NODES collection with one circuit visiting once all vertices of G.

Arc input(s) NODES

Arc generator $CLIQUE \mapsto$ collection(nodes1, nodes2)

Arc arity 2

Arc constraint(s) nodes1.succ = nodes2.index

Graph property(ies)
- $\mathbf{MIN\_NSCC} = |\texttt{NODES}|$
- $\mathbf{MAX\_ID} = 1$

Example

$$\texttt{circuit}\left(\left\{\begin{array}{ll}\texttt{index}-1 & \texttt{succ}-2,\\ \texttt{index}-2 & \texttt{succ}-3,\\ \texttt{index}-3 & \texttt{succ}-4,\\ \texttt{index}-4 & \texttt{succ}-1\end{array}\right\}\right)$$

Parts (A) and (B) of Figure 4.92 respectively show the initial and final graph. The circuit constraint holds since the final graph consists of one circuit mentioning once every vertex of the initial graph.

Graph model The first graph property enforces to have one single strongly connected component containing |NODES| 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 |NODES| vertices the final graph contains at most |NODES| vertices. Therefore we can rewrite the graph property $\mathbf{MIN\_NSCC} = |\texttt{NODES}|$ to $\mathbf{MIN\_NSCC} \geq |\texttt{NODES}|$. This leads to simplify $\overline{\mathbf{MIN\_NSCC}}$ to $\overline{\mathbf{MIN\_NSCC}}$.

Because of the graph property $\mathbf{MIN\_NSCC} = |\texttt{NODES}|$ 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 $\mathbf{MAX\_ID}$ is greater than or equal to 1. So we can rewrite the graph property $\mathbf{MAX\_ID} = 1$ to $\mathbf{MAX\_ID} \leq 1$. This leads to simplify $\overline{\underline{\mathbf{MAX\_ID}}}$ to $\underline{\mathbf{MAX\_ID}}$.

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.

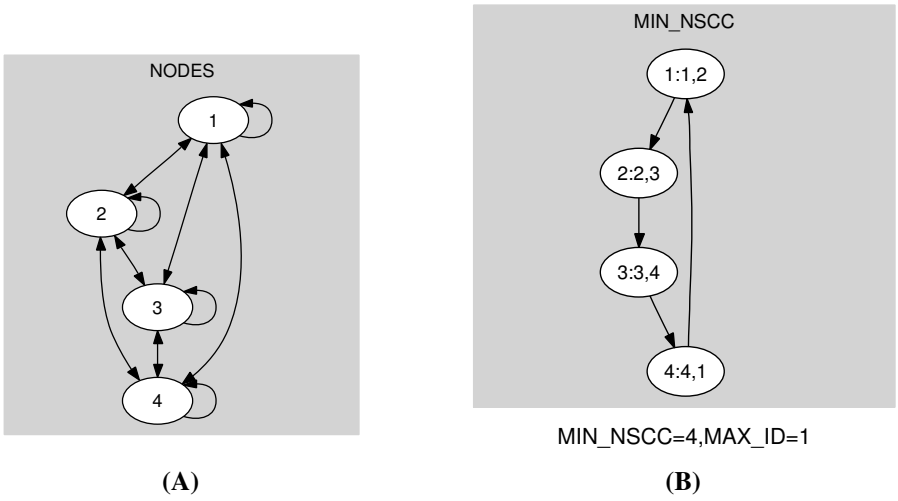


Figure 4.92: Initial and final graph of the `circuit` constraint

4.39 circuit_cluster

Origin Inspired by [77].

Constraint circuit_cluster(NCIRCUIT, NODES)

Argument(s)

NCIRCUIT	:	dvar
NODES	:	collection(index − int, cluster − int, succ − dvar)

Restriction(s)
- NCIRCUIT ≥ 1
- NCIRCUIT $\leq$ |NODES|
- required(NODES, [index, cluster, succ])
- NODES.index ≥ 1
- NODES.index $\leq$ |NODES|
- distinct(NODES, index)
- NODES.succ ≥ 1
- NODES.succ $\leq$ |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.

Arc input(s) NODES

Arc generator $CLIQUE \mapsto$ collection(nodes1, nodes2)

Arc arity 2

Arc constraint(s)
- nodes1.succ $\neq$ nodes1.index
- nodes1.succ $=$ nodes2.index

Graph property(ies)
- $\mathbf{NTREE} = 0$
- $\mathbf{NSCC} =$ NCIRCUIT

Sets

$$\text{ALL\_VERTICES} \mapsto \left[\text{variables} - \text{col}\left(\begin{array}{l} \text{VARIABLES} - \text{collection}(\text{var} - \text{dvar}), \\ [\text{item}(\text{var} - \text{NODES.cluster})] \end{array} \right) \right]$$

Constraint(s) on sets
- alldifferent(variables)
- nvalues(variables, =, size(NODES, cluster))

Example

$$\texttt{circuit\_cluster}\left(1,\left\{\begin{array}{lll}\texttt{index}-1 & \texttt{cluster}-1 & \texttt{succ}-1,\\ \texttt{index}-2 & \texttt{cluster}-1 & \texttt{succ}-4,\\ \texttt{index}-3 & \texttt{cluster}-2 & \texttt{succ}-3,\\ \texttt{index}-4 & \texttt{cluster}-2 & \texttt{succ}-5,\\ \texttt{index}-5 & \texttt{cluster}-3 & \texttt{succ}-8,\\ \texttt{index}-6 & \texttt{cluster}-3 & \texttt{succ}-6,\\ \texttt{index}-7 & \texttt{cluster}-3 & \texttt{succ}-7,\\ \texttt{index}-8 & \texttt{cluster}-4 & \texttt{succ}-2,\\ \texttt{index}-9 & \texttt{cluster}-4 & \texttt{succ}-9\end{array}\right\}\right)$$

$$\texttt{circuit\_cluster}\left(2,\left\{\begin{array}{lll}\texttt{index}-1 & \texttt{cluster}-1 & \texttt{succ}-1,\\ \texttt{index}-2 & \texttt{cluster}-1 & \texttt{succ}-4,\\ \texttt{index}-3 & \texttt{cluster}-2 & \texttt{succ}-3,\\ \texttt{index}-4 & \texttt{cluster}-2 & \texttt{succ}-2,\\ \texttt{index}-5 & \texttt{cluster}-3 & \texttt{succ}-5,\\ \texttt{index}-6 & \texttt{cluster}-3 & \texttt{succ}-9,\\ \texttt{index}-7 & \texttt{cluster}-3 & \texttt{succ}-7,\\ \texttt{index}-8 & \texttt{cluster}-4 & \texttt{succ}-8,\\ \texttt{index}-9 & \texttt{cluster}-4 & \texttt{succ}-6\end{array}\right\}\right)$$

Parts (A) and (B) of Figure 4.93 respectively show the initial and final graph associated to the second example. Since we use the $\mathbf{NSCC}$ 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 $\mathbf{NTREE} = 0$. The first example uses only one single circuit: $2 \rightarrow 4 \rightarrow 5 \rightarrow 8 \rightarrow 2$.

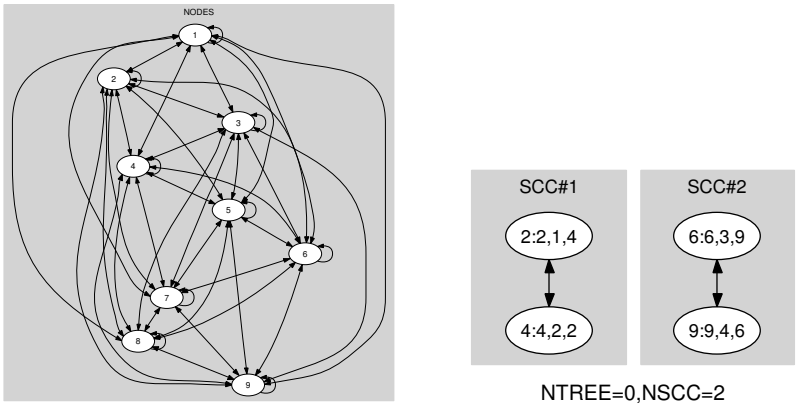


Figure 4.93: Initial and final graph of the `circuit_cluster` constraint

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 $\neq$ 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 $\mathbf{NTREE} = 0$ enforces that all vertices of the final graph belong to one circuit.
- The graph property $\mathbf{NSCC}$ $=$ 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.

4.40 circular_change

Origin Derived from change.

Constraint circular_change(NCHANGE, VARIABLES, CTR)

Argument(s)

NCHANGE	:	dvar
VARIABLES	:	collection(var − dvar)
CTR	:	atom

Restriction(s)

NCHANGE ≥ 0
NCHANGE $\leq$ |VARIABLES|
required(VARIABLES, var)
CTR $\in [=, \neq, <, \geq, >, \leq]$

Purpose 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.

Arc input(s) VARIABLES

Arc generator $CIRCUIT \mapsto$ collection(variables1, variables2)

Arc arity 2

Arc constraint(s) variables1.var CTR variables2.var

Graph property(ies) $\mathbf{NARC} =$ NCHANGE

Example

$$\texttt{circular\_change}\left(\begin{array}{l} 4, \left\{\begin{array}{l} \texttt{var}-4,\\ \texttt{var}-4,\\ \texttt{var}-3,\\ \texttt{var}-4,\\ \texttt{var}-1 \end{array}\right\}, \neq \end{array}\right)$$

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 consecutives 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.

Graph model Since we are also interested in the constraint that links the last and the first variable we use the arc generator $CIRCUIT$ to produce the arcs of the initial graph.

Automaton Figure 4.95 depicts the automaton associated to the circular_change constraint. To each pair of consecutive variables $(\texttt{VAR}_i, \texttt{VAR}_{(i \bmod |\texttt{VARIABLES}|)+1})$ of the collection VARIABLES corresponds a 0-1 signature variable $\texttt{S}_i$. The following signature constraint links $\texttt{VAR}_i$, $\texttt{VAR}_{(i \bmod |\texttt{VARIABLES}|)+1}$ and $\texttt{S}_i$: $\texttt{VAR}_i$ CTR $\texttt{VAR}_{(i \bmod |\texttt{VARIABLES}|)+1} \Leftrightarrow \texttt{S}_i$.

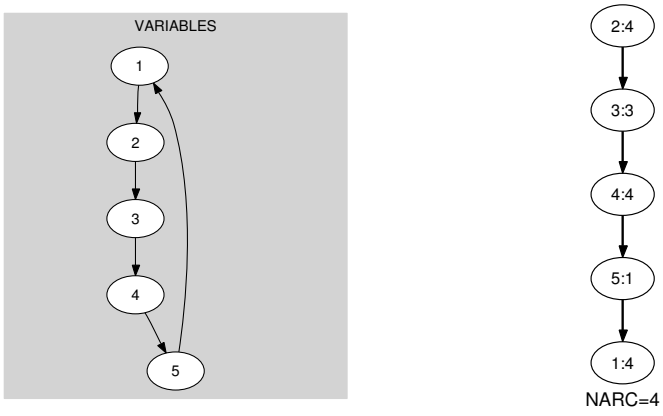


Figure 4.94: Initial and final graph of the `circular_change` constraint

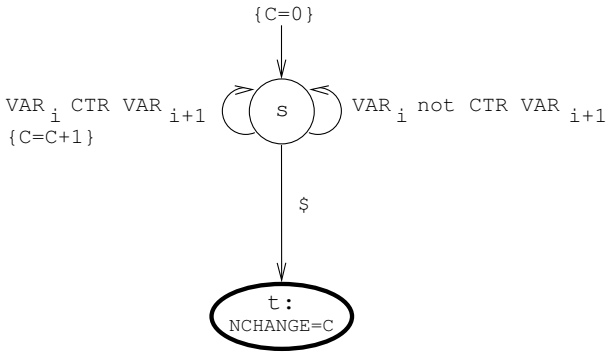


Figure 4.95: Automaton of the `circular_change` constraint

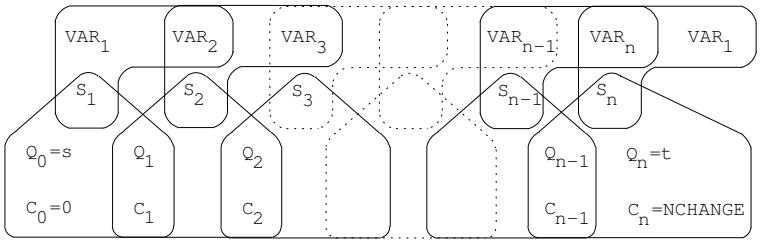


Figure 4.96: Hypergraph of the reformulation corresponding to the automaton of the `circular_change` constraint

See also `change`.

Key words timetabling constraint, number of changes, cyclic, automaton, automaton with counters, circular sliding cyclic(1) constraint network(2).

4.41 clique

Origin [78]

Constraint clique(SIZE_CLIQUE, NODES)

Argument(s)

SIZE_CLIQUE	:	dvar
NODES	:	collection(index − int, succ − svar)

Restriction(s)

SIZE_CLIQUE ≥ 0
SIZE_CLIQUE $\leq |$NODES$|$
required(NODES, [index, succ])
NODES.index ≥ 1
NODES.index $\leq |$NODES$|$
distinct(NODES, index)

Purpose

Consider a digraph G described by the NODES collection: To the i^{th} item of the NODES collection corresponds the i^{th} vertex of G; To each value j of the i^{th} succ 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 SIZE_CLIQUE (i.e. there is an arc between each pair of distinct vertices of $\mathcal{S}$).

Arc input(s) NODES

Arc generator $CLIQUE(\neq) \mapsto$ collection(nodes1, nodes2)

Arc arity 2

Arc constraint(s) in_set(nodes2.index, nodes1.succ)

Graph property(ies)

- $\mathbf{NARC}$ = SIZE_CLIQUE $*$ SIZE_CLIQUE − SIZE_CLIQUE
- $\mathbf{NVERTEX}$ = SIZE_CLIQUE

Example

$$\texttt{clique}\left(\begin{array}{l} 3, \left\{\begin{array}{ll} \texttt{index} - 1 & \texttt{succ} - \emptyset, \\ \texttt{index} - 2 & \texttt{succ} - \{3,5\}, \\ \texttt{index} - 3 & \texttt{succ} - \{2,5\}, \\ \texttt{index} - 4 & \texttt{succ} - \emptyset, \\ \texttt{index} - 5 & \texttt{succ} - \{2,3\} \end{array}\right\} \end{array}\right)$$

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 succ attribute of a given vertex. Part (B) of Figure 4.97 gives the final graph associated to the example. Since we both use the $\mathbf{NARC}$ and $\mathbf{NVERTEX}$ 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.

Graph model Observe the use of *set variables* 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.

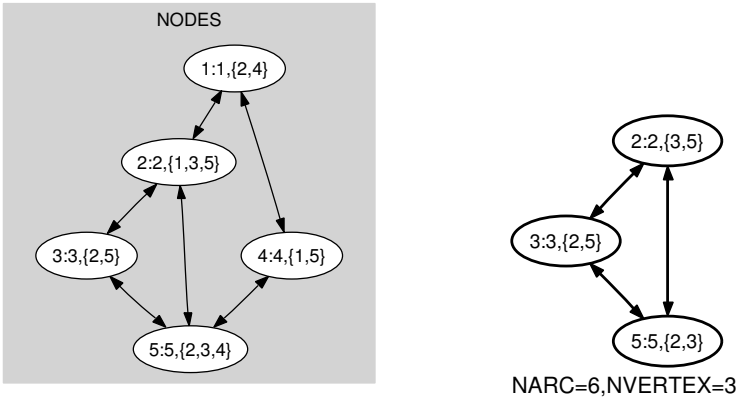


Figure 4.97: Initial and final graph of the `clique` set constraint

4.42 colored_matrix

Origin KOALOG

Constraint colored_matrix(C, L, K, MATRIX, CPROJ, LPROJ)

Synonym(s) cardinality_matrix, card_matrix.

Argument(s)

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)

Restriction(s)

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

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

$$\texttt{colored\_matrix}\left(\begin{array}{l} 1,2,4,\left\{\begin{array}{lll}\texttt{column}-0 & \texttt{line}-0 & \texttt{var}-3,\\ \texttt{column}-0 & \texttt{line}-1 & \texttt{var}-1,\\ \texttt{column}-0 & \texttt{line}-2 & \texttt{var}-3,\\ \texttt{column}-1 & \texttt{line}-0 & \texttt{var}-4,\\ \texttt{column}-1 & \texttt{line}-1 & \texttt{var}-4,\\ \texttt{column}-1 & \texttt{line}-2 & \texttt{var}-3\end{array}\right\},\\ \left\{\begin{array}{lll}\texttt{column}-0 & \texttt{val}-0 & \texttt{nocc}-0,\\ \texttt{column}-0 & \texttt{val}-1 & \texttt{nocc}-1,\\ \texttt{column}-0 & \texttt{val}-2 & \texttt{nocc}-0,\\ \texttt{column}-0 & \texttt{val}-3 & \texttt{nocc}-2,\\ \texttt{column}-0 & \texttt{val}-4 & \texttt{nocc}-0,\\ \texttt{column}-1 & \texttt{val}-0 & \texttt{nocc}-0,\\ \texttt{column}-1 & \texttt{val}-1 & \texttt{nocc}-0,\\ \texttt{column}-1 & \texttt{val}-2 & \texttt{nocc}-0,\\ \texttt{column}-1 & \texttt{val}-3 & \texttt{nocc}-1,\\ \texttt{column}-1 & \texttt{val}-4 & \texttt{nocc}-2\end{array}\right\},\\ \left\{\begin{array}{lll}\texttt{line}-0 & \texttt{val}-0 & \texttt{nocc}-0,\\ \texttt{line}-0 & \texttt{val}-1 & \texttt{nocc}-0,\\ \texttt{line}-0 & \texttt{val}-2 & \texttt{nocc}-0,\\ \texttt{line}-0 & \texttt{val}-3 & \texttt{nocc}-1,\\ \texttt{line}-0 & \texttt{val}-4 & \texttt{nocc}-1,\\ \texttt{line}-1 & \texttt{val}-0 & \texttt{nocc}-0,\\ \texttt{line}-1 & \texttt{val}-1 & \texttt{nocc}-1,\\ \texttt{line}-1 & \texttt{val}-2 & \texttt{nocc}-0,\\ \texttt{line}-1 & \texttt{val}-3 & \texttt{nocc}-0,\\ \texttt{line}-1 & \texttt{val}-4 & \texttt{nocc}-1,\\ \texttt{line}-2 & \texttt{val}-0 & \texttt{nocc}-0,\\ \texttt{line}-2 & \texttt{val}-1 & \texttt{nocc}-0,\\ \texttt{line}-2 & \texttt{val}-2 & \texttt{nocc}-0,\\ \texttt{line}-2 & \texttt{val}-3 & \texttt{nocc}-2,\\ \texttt{line}-2 & \texttt{val}-4 & \texttt{nocc}-0\end{array}\right\}\end{array}\right)$$

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 cumulative and nvalues.

Constraint coloured_cumulative(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 coloured_cumulative 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 origin + duration = end.

Arc input(s) TASKS

Arc generator $SELF \mapsto$ collection(tasks)

Arc arity 1

Arc constraint(s) tasks.origin + tasks.duration = tasks.end

Graph property(ies) $\mathbf{NARC} = |\text{TASKS}|$

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 $<$ tasks2.end

Sets

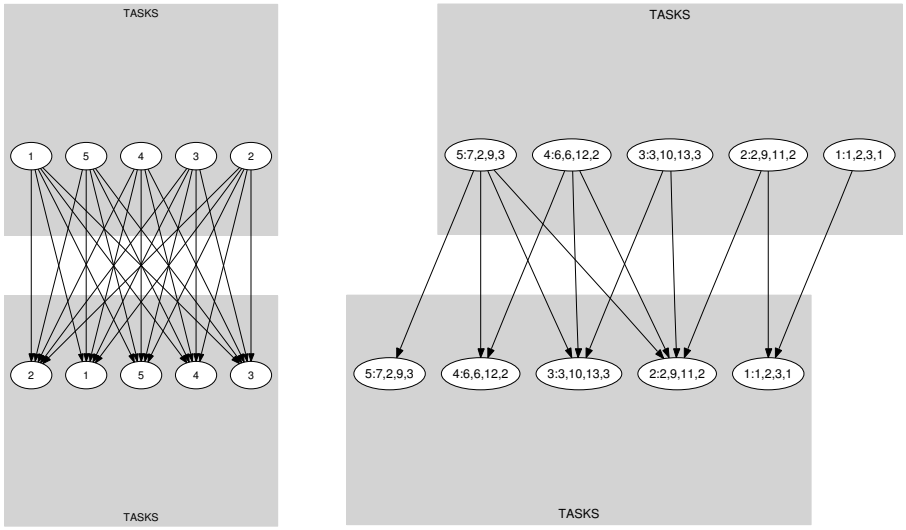
$$\text{SUCC} \mapsto \left[\begin{array}{l} \text{source}, \\ \text{variables} - \text{col}\left(\begin{array}{l} \text{VARIABLES} - \text{collection}(\text{var} - \text{dvar}), \\ [\text{item}(\text{var} - \text{TASKS.colour})] \end{array}\right) \end{array}\right]$$

Constraint(s) on sets nvalues(variables, $\leq$, LIMIT)

Example

$$\texttt{coloured\_cumulative}\left(\left\{\begin{array}{llll} \texttt{origin}-1 & \texttt{duration}-2 & \texttt{end}-3 & \texttt{colour}-1,\\ \texttt{origin}-2 & \texttt{duration}-9 & \texttt{end}-11 & \texttt{colour}-2,\\ \texttt{origin}-3 & \texttt{duration}-10 & \texttt{end}-13 & \texttt{colour}-3,\\ \texttt{origin}-6 & \texttt{duration}-6 & \texttt{end}-12 & \texttt{colour}-2,\\ \texttt{origin}-7 & \texttt{duration}-2 & \texttt{end}-9 & \texttt{colour}-3 \end{array}\right\}, 2\right)$$

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.



(A) **(B)**

Figure 4.98: Initial and final graph of the `coloured_cumulative` constraint

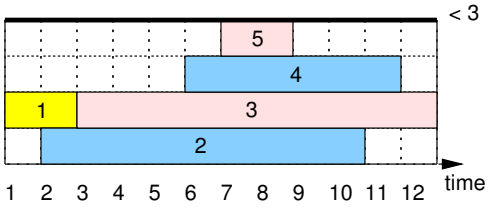


Figure 4.99: A coloured cumulative solution with at most two distinct colours in parallel

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 $\mathbf{NARC} = |\texttt{TASKS}|$ to $\mathbf{NARC} \geq |\texttt{TASKS}|$. This leads to simplify $\underline{\overline{\mathbf{NARC}}}$ to $\overline{\mathbf{NARC}}$.

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.

4.44 coloured_cumulatives

Origin Derived from cumulatives and nvalues.

Constraint coloured_cumulatives(TASKS, MACHINES)

Argument(s)

TASKS	:	collection	$\left(\begin{array}{l}\texttt{machine} - \texttt{dvar},\\ \texttt{origin} - \texttt{dvar},\\ \texttt{duration} - \texttt{dvar},\\ \texttt{end} - \texttt{dvar},\\ \texttt{colour} - \texttt{dvar}\end{array}\right)$
MACHINES	:	collection(id − int, capacity − int)	

Restriction(s)
- 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

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 origin + duration = end.

Arc input(s) TASKS

Arc generator $SELF \mapsto$ collection(tasks)

Arc arity 1

Arc constraint(s) tasks.origin + tasks.duration = tasks.end

Graph property(ies) $\mathbf{NARC} = |$TASKS$|$

For all items of MACHINES:

Arc input(s) TASKS TASKS

Arc generator $PRODUCT \mapsto$ collection(tasks1, tasks2)

Arc arity 2

Arc constraint(s)
- tasks1.machine = MACHINES.id
- tasks1.machine = tasks2.machine
- tasks1.duration > 0
- tasks2.origin $\leq$ tasks1.origin
- tasks1.origin $<$ tasks2.end

Sets

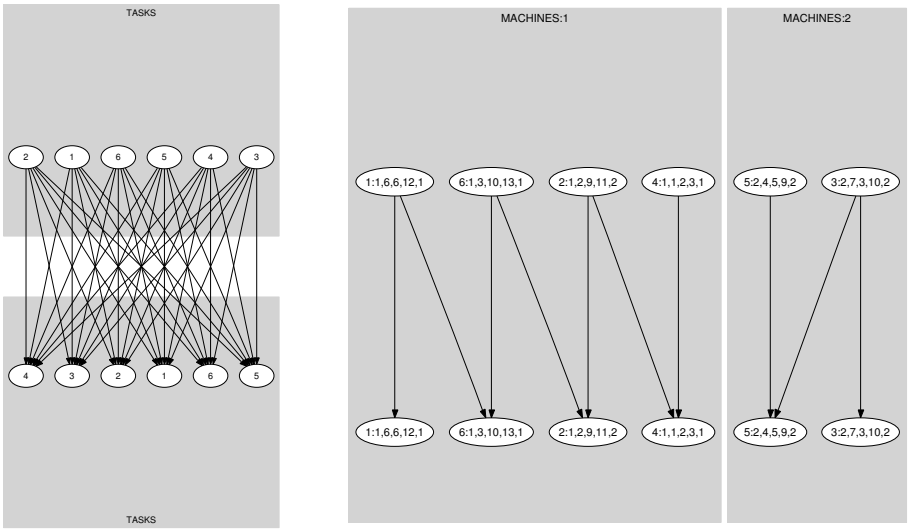
$$\text{SUCC} \mapsto \left[\begin{array}{l} \texttt{source}, \\ \texttt{variables} - \texttt{col} \left(\begin{array}{l} \texttt{VARIABLES} - \texttt{collection}(\texttt{var} - \texttt{dvar}), \\ [\texttt{item}(\texttt{var} - \texttt{TASKS.colour})] \end{array} \right) \end{array} \right]$$

Constraint(s) on sets $\texttt{nvalues}(\texttt{variables}, \leq, \texttt{MACHINES.capacity})$

Example

$$\texttt{coloured\_cumulatives} \left(\begin{array}{l} \left\{ \begin{array}{lllll} \texttt{machine} - 1 & \texttt{origin} - 6 & \texttt{duration} - 6 & \texttt{end} - 12 & \texttt{colour} - 1, \\ \texttt{machine} - 1 & \texttt{origin} - 2 & \texttt{duration} - 9 & \texttt{end} - 11 & \texttt{colour} - 2, \\ \texttt{machine} - 2 & \texttt{origin} - 7 & \texttt{duration} - 3 & \texttt{end} - 10 & \texttt{colour} - 2, \\ \texttt{machine} - 1 & \texttt{origin} - 1 & \texttt{duration} - 2 & \texttt{end} - 3 & \texttt{colour} - 1, \\ \texttt{machine} - 2 & \texttt{origin} - 4 & \texttt{duration} - 5 & \texttt{end} - 9 & \texttt{colour} - 2, \\ \texttt{machine} - 1 & \texttt{origin} - 3 & \texttt{duration} - 10 & \texttt{end} - 13 & \texttt{colour} - 1 \end{array} \right), \\ \left\{ \begin{array}{ll} \texttt{id} - 1 & \texttt{capacity} - 2, \\ \texttt{id} - 2 & \texttt{capacity} - 1 \end{array} \right\} \end{array} \right)$$

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.



(A) **(B)**

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 $\mathbf{NARC} = |\texttt{TASKS}|$ to $\mathbf{NARC} \geq |\texttt{TASKS}|$. This leads to simplify $\underline{\overline{\mathbf{NARC}}}$ to $\overline{\mathbf{NARC}}$.

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 coloured_cumulative, cumulative, cumulatives, nvalues.

Key words scheduling constraint, resource constraint, temporal constraint, coloured, number of distinct values.

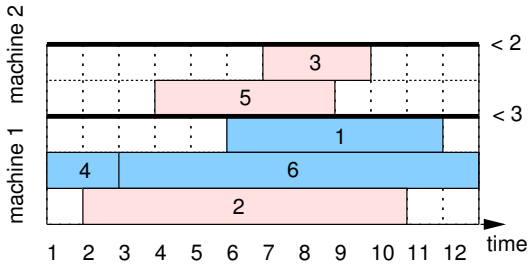


Figure 4.101: Assignment of the tasks on the two machines

4.45 common

Origin N. Beldiceanu

Constraint common(NCOMMON1, NCOMMON2, VARIABLES1, VARIABLES2)

Argument(s)

NCOMMON1	:	dvar
NCOMMON2	:	dvar
VARIABLES1	:	collection(var − dvar)
VARIABLES2	:	collection(var − dvar)

Restriction(s)

NCOMMON1 ≥ 0
NCOMMON1 $\leq |$VARIABLES1$|$
NCOMMON2 ≥ 0
NCOMMON2 $\leq |$VARIABLES2$|$
required(VARIABLES1, var)
required(VARIABLES2, var)

Purpose

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.

Arc input(s) VARIABLES1 VARIABLES2

Arc generator $PRODUCT \mapsto$ collection(variables1, variables2)

Arc arity 2

Arc constraint(s) variables1.var = variables2.var

Graph property(ies)
- $\mathbf{NSOURCE}$ = NCOMMON1
- $\mathbf{NSINK}$ = NCOMMON2

Example

$$\texttt{common}\left(\begin{array}{l} 3, 4, \{\texttt{var}-1, \texttt{var}-9, \texttt{var}-1, \texttt{var}-5\}, \\ \left\{\begin{array}{l} \texttt{var}-2, \\ \texttt{var}-1, \\ \texttt{var}-9, \\ \texttt{var}-9, \\ \texttt{var}-6, \\ \texttt{var}-9 \end{array}\right\} \end{array}\right)$$

Parts (A) and (B) of Figure 4.102 respectively show the initial and final graph. Since we use the $\mathbf{NSOURCE}$ and $\mathbf{NSINK}$ 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.

See also alldifferent_on_intersection, nvalue_on_intersection, same_intersection.

Key words constraint between two collections of variables, acyclic, bipartite, no_loop.

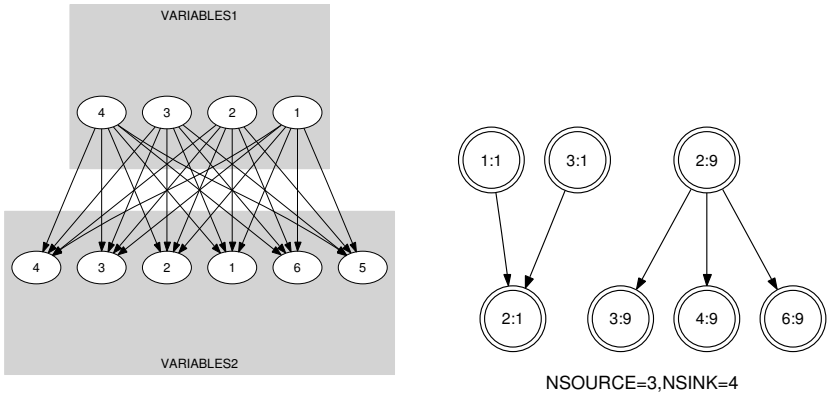


Figure 4.102: Initial and final graph of the common constraint

4.46 common_interval

Origin Derived from common.

Constraint common_interval(NCOMMON1, NCOMMON2, VARIABLES1, VARIABLES2, SIZE_INTERVAL)

Argument(s)

NCOMMON1	:	dvar
NCOMMON2	:	dvar
VARIABLES1	:	collection(var − dvar)
VARIABLES2	:	collection(var − dvar)
SIZE_INTERVAL	:	int

Restriction(s)

NCOMMON1 ≥ 0
NCOMMON1 $\leq |$VARIABLES1$|$
NCOMMON2 ≥ 0
NCOMMON2 $\leq |$VARIABLES2$|$
required(VARIABLES1, var)
required(VARIABLES2, var)
SIZE_INTERVAL > 0

Purpose

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 $- 1]$.
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 $- 1]$.

Arc input(s) VARIABLES1 VARIABLES2

Arc generator $PRODUCT \mapsto$ collection(variables1, variables2)

Arc arity 2

Arc constraint(s) variables1.var/SIZE_INTERVAL = variables2.var/SIZE_INTERVAL

Graph property(ies)

- **NSOURCE** = NCOMMON1
- **NSINK** = NCOMMON2

Example

$$\texttt{common\_interval}\left(\left\{\begin{array}{l}3,2,\left\{\begin{array}{l}\texttt{var}-8,\\ \texttt{var}-6,\\ \texttt{var}-6,\\ \texttt{var}-0\end{array}\right\},\\ \left\{\begin{array}{l}\texttt{var}-7,\\ \texttt{var}-3,\\ \texttt{var}-3,\\ \texttt{var}-3,\\ \texttt{var}-3,\\ \texttt{var}-7\end{array}\right\},3\end{array}\right\}\right)$$

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.

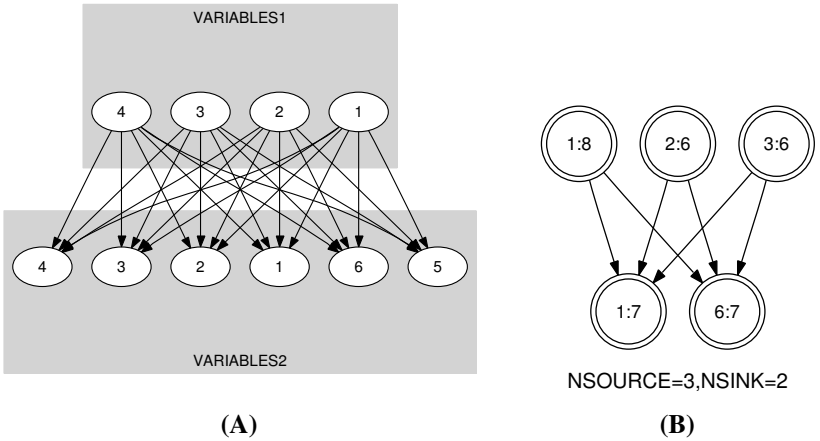


Figure 4.103: Initial and final graph of the common_interval constraint

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 common_modulo(NCOMMON1, NCOMMON2, VARIABLES1, VARIABLES2, M)

Argument(s)

NCOMMON1	:	dvar
NCOMMON2	:	dvar
VARIABLES1	:	collection(var − dvar)
VARIABLES2	:	collection(var − dvar)
M	:	int

Restriction(s)

NCOMMON1 ≥ 0
NCOMMON1 $\leq |$VARIABLES1$|$
NCOMMON2 ≥ 0
NCOMMON2 $\leq |$VARIABLES2$|$
required(VARIABLES1, var)
required(VARIABLES2, var)
M > 0

Purpose

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.
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.

Arc input(s) VARIABLES1 VARIABLES2

Arc generator $PRODUCT \mapsto$ collection(variables1, variables2)

Arc arity 2

Arc constraint(s) variables1.var $\bmod$ M $=$ variables2.var $\bmod$ M

Graph property(ies)
- **NSOURCE** $=$ NCOMMON1
- **NSINK** $=$ NCOMMON2

Example

$$\texttt{common\_modulo}\left(\begin{array}{l} 3, 4, \{\texttt{var}-0, \texttt{var}-4, \texttt{var}-0, \texttt{var}-8\}, \\ \left\{\begin{array}{l} \texttt{var}-7, \\ \texttt{var}-5, \\ \texttt{var}-4, \\ \texttt{var}-9, \\ \texttt{var}-2, \\ \texttt{var}-4 \end{array}\right\}, 5 \end{array}\right)$$

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.

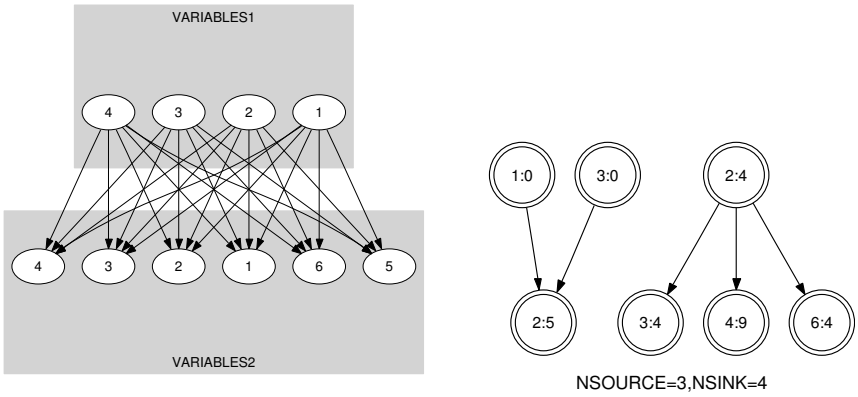


Figure 4.104: Initial and final graph of the common_modulo constraint

See also common.

Key words constraint between two collections of variables, modulo, acyclic, bipartite, no_loop.

4.48 common_partition

Origin Derived from common.

Constraint common_partition(NCOMMON1, NCOMMON2, VARIABLES1, VARIABLES2, PARTITIONS)

Type(s) VALUES : collection(val − int)

Argument(s)

NCOMMON1	:	dvar
NCOMMON2	:	dvar
VARIABLES1	:	collection(var − dvar)
VARIABLES2	:	collection(var − dvar)
PARTITIONS	:	collection(p − VALUES)

Restriction(s)
required(VALUES, val)
distinct(VALUES, val)
NCOMMON1 ≥ 0
NCOMMON1 $\leq$ |VARIABLES1|
NCOMMON2 ≥ 0
NCOMMON2 $\leq$ |VARIABLES2|
required(VARIABLES1, var)
required(VARIABLES2, var)
required(PARTITIONS, p)
|PARTITIONS| ≥ 2

Purpose
NCOMMON1 is the number of variables of the VARIABLES1 collection taking a value in a partition derived from the values assigned to the variables of VARIABLES2 and from PARTITIONS.
NCOMMON2 is the number of variables of the VARIABLES2 collection taking a value in a partition derived from the values assigned to the variables of VARIABLES1 and from PARTITIONS.

Arc input(s) VARIABLES1 VARIABLES2

Arc generator $PRODUCT \mapsto$ collection(variables1, variables2)

Arc arity 2

Arc constraint(s) in_same_partition(variables1.var, variables2.var, PARTITIONS)

Graph property(ies)
- $\mathbf{NSOURCE}$ = NCOMMON1
- $\mathbf{NSINK}$ = NCOMMON2

Example

$$\texttt{common\_partition}\left(\begin{array}{l} 3, 4, \left\{\begin{array}{l} \texttt{var}-2,\\ \texttt{var}-3,\\ \texttt{var}-6,\\ \texttt{var}-0 \end{array}\right\},\\ \left\{\begin{array}{l} \texttt{var}-0,\\ \texttt{var}-6,\\ \texttt{var}-3,\\ \texttt{var}-3,\\ \texttt{var}-7,\\ \texttt{var}-1 \end{array}\right\},\\ \left\{\begin{array}{l} \texttt{p}-\{\texttt{val}-1, \texttt{val}-3\},\\ \texttt{p}-\{\texttt{val}-4\},\\ \texttt{p}-\{\texttt{val}-2, \texttt{val}-6\} \end{array}\right\} \end{array}\right)$$

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.

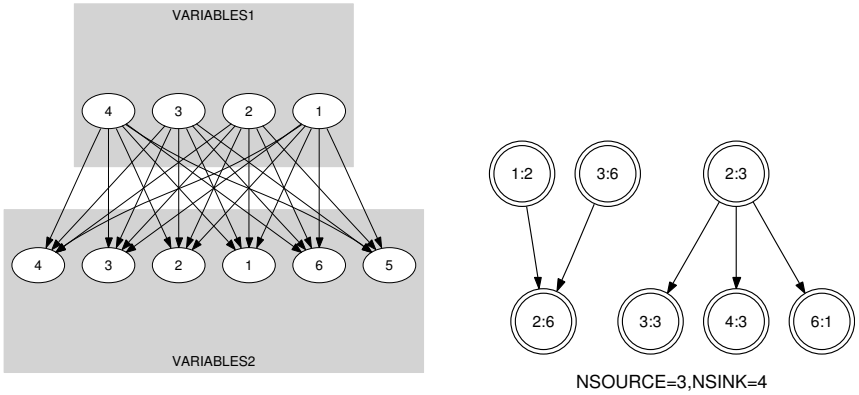


Figure 4.105: Initial and final graph of the common_partition constraint

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 connect_points(SIZE1, SIZE2, SIZE3, NGROUP, POINTS)

Argument(s)

SIZE1	:	int
SIZE2	:	int
SIZE3	:	int
NGROUP	:	dvar
POINTS	:	collection(p − dvar)

Restriction(s)

SIZE1 > 0
SIZE2 > 0
SIZE3 > 0
NGROUP ≥ 0
NGROUP $\leq |$POINTS$|$
SIZE1 $*$ SIZE2 $*$ SIZE3 $=$ |POINTS|
required(POINTS, p)

Purpose 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.

Arc input(s) POINTS

Arc generator $GRID([\texttt{SIZE1}, \texttt{SIZE2}, \texttt{SIZE3}]) \mapsto \texttt{collection}(\texttt{points1}, \texttt{points2})$

Arc arity 2

Arc constraint(s)

- points1.p $\neq 0$
- points1.p $=$ points2.p

Graph property(ies) $\mathbf{NSCC} =$ NGROUP

Example

$$\texttt{connect\_points}\left(\begin{array}{l} 8,4,2,2,\left\{\begin{array}{l} \mathtt{p}-0,\mathtt{p}-0,\\ \mathtt{p}-1,\mathtt{p}-1,\\ \mathtt{p}-0,\mathtt{p}-2,\\ \mathtt{p}-0,\mathtt{p}-0,\\ \mathtt{p}-0,\mathtt{p}-0,\\ \mathtt{p}-0,\mathtt{p}-1,\\ \mathtt{p}-0,\mathtt{p}-2,\\ \mathtt{p}-0,\mathtt{p}-0,\\ \mathtt{p}-0,\mathtt{p}-0,\\ \mathtt{p}-0,\mathtt{p}-1,\\ \mathtt{p}-1,\mathtt{p}-1,\\ \mathtt{p}-1,\mathtt{p}-1,\\ \mathtt{p}-0,\mathtt{p}-2,\\ \mathtt{p}-0,\mathtt{p}-1,\\ \mathtt{p}-0,\mathtt{p}-2,\\ \mathtt{p}-0,\mathtt{p}-0,\\ \mathtt{p}-0,\mathtt{p}-0,\\ \mathtt{p}-0,\mathtt{p}-0,\\ \mathtt{p}-0,\mathtt{p}-0,\\ \mathtt{p}-0,\mathtt{p}-0,\\ \mathtt{p}-0,\mathtt{p}-0,\\ \mathtt{p}-0,\mathtt{p}-2,\\ \mathtt{p}-0,\mathtt{p}-0,\\ \mathtt{p}-0,\mathtt{p}-2,\\ \mathtt{p}-2,\mathtt{p}-2,\\ \mathtt{p}-2,\mathtt{p}-2,\\ \mathtt{p}-0,\mathtt{p}-0,\\ \mathtt{p}-0,\mathtt{p}-2,\\ \mathtt{p}-0,\mathtt{p}-0,\\ \mathtt{p}-0,\mathtt{p}-2,\\ \mathtt{p}-0,\mathtt{p}-0 \end{array}\right\} \end{array}\right)$$

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.

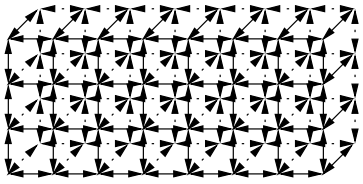


Figure 4.106: Graph generated by *GRID* `([8,4,2])`

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	**1**	0	**2**	0	0
0	0	0	**1**	**1**	**1**	**1**	**1**
0	**2**	0	**1**	0	**2**	0	0

0	0	0	0	0	0	0	0
0	0	0	0	0	**2**	0	0
0	**2**	**2**	**2**	**2**	**2**	0	0
0	**2**	0	0	0	**2**	0	0

Figure 4.107: The two layers of the solution

4.50 correspondence

Origin Derived from sort_permutation by removing the sorting condition.

Constraint correspondence(FROM, PERMUTATION, TO)

Argument(s)

FROM	:	collection(fvar − dvar)
PERMUTATION	:	collection(var − dvar)
TO	:	collection(tvar − dvar)

Restriction(s)

|PERMUTATION| = |FROM|
|PERMUTATION| = |TO|
PERMUTATION.var ≥ 1
PERMUTATION.var ≤ |PERMUTATION|
alldifferent(PERMUTATION)
required(FROM, fvar)
required(PERMUTATION, var)
required(TO, tvar)

Purpose The variables of the TO collection correspond to the variables of the FROM collection according to the permutation expressed by PERMUTATION.

Derived Collection(s)

$$\texttt{col}\left(\begin{array}{l}\texttt{FROM\_PERMUTATION} - \texttt{collection}(\texttt{fvar} - \texttt{dvar}, \texttt{var} - \texttt{dvar}),\\ [\texttt{item}(\texttt{fvar} - \texttt{FROM.fvar}, \texttt{var} - \texttt{PERMUTATION.var})]\end{array}\right)$$

Arc input(s) FROM_PERMUTATION TO

Arc generator $PRODUCT \mapsto$ collection(from_permutation, to)

Arc arity 2

Arc constraint(s)

- from_permutation.fvar = to.tvar
- from_permutation.var = to.key

Graph property(ies) NARC = |PERMUTATION|

Example

$$\texttt{correspondence}\left(\left\{\begin{array}{l}\texttt{fvar}-1,\\ \texttt{fvar}-9,\\ \texttt{fvar}-1,\\ \texttt{fvar}-5,\\ \texttt{fvar}-2,\\ \texttt{fvar}-1\end{array}\right\},\left\{\begin{array}{l}\texttt{var}-6,\\ \texttt{var}-1,\\ \texttt{var}-3,\\ \texttt{var}-5,\\ \texttt{var}-4,\\ \texttt{var}-2\end{array}\right\},\left\{\begin{array}{l}\texttt{tvar}-9,\\ \texttt{tvar}-1,\\ \texttt{tvar}-1,\\ \texttt{tvar}-2,\\ \texttt{tvar}-5,\\ \texttt{tvar}-1\end{array}\right\}\right)$$

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.

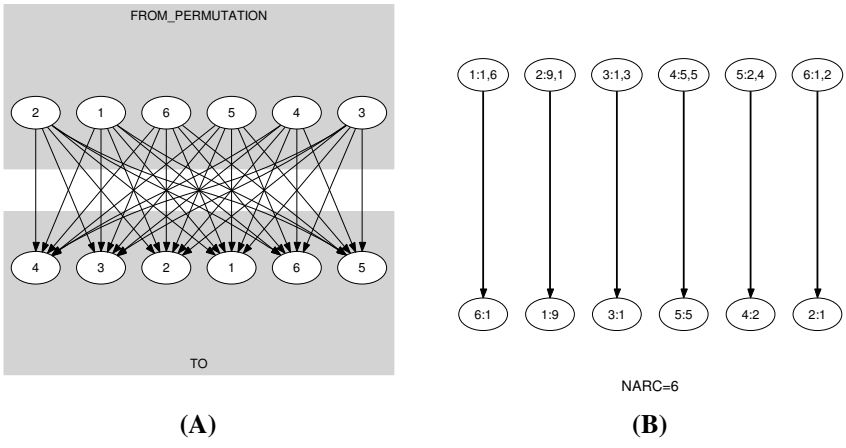


Figure 4.108: Initial and final graph of the correspondence constraint

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 $\mathbf{NARC} = |\texttt{PERMUTATION}|$ to $\mathbf{NARC} \geq |\texttt{PERMUTATION}|$. This leads to simplify $\overline{\mathbf{NARC}}$ to $\overline{\mathbf{NARC}}$.

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.

4.51 count

Origin [46]

Constraint count(VALUE, VARIABLES, RELOP, NVAR)

Argument(s)

VALUE	:	int
VARIABLES	:	collection(var − dvar)
RELOP	:	atom
NVAR	:	dvar

Restriction(s) required(VARIABLES, var)
RELOP $\in [=, \neq, <, \geq, >, \leq]$

Purpose Let N be the number of variables of the VARIABLES collection assigned to value VAL; Enforce condition N RELOP NVAR to hold.

Arc input(s) VARIABLES

Arc generator $SELF \mapsto$ collection(variables)

Arc arity 1

Arc constraint(s) variables.var = VALUE

Graph property(ies) **NARC** RELOP NVAR

Example

$$\texttt{count}\left(\left\langle 5, \left\{\begin{array}{l}\texttt{var}-4,\\ \texttt{var}-5,\\ \texttt{var}-5,\\ \texttt{var}-4,\\ \texttt{var}-5\end{array}\right\}, \geq, 2\right\rangle\right)$$

The constraint holds since value 5 occurs 3 times, which is greater than or equal to 2. Parts (A) and (B) of Figure 4.109 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.

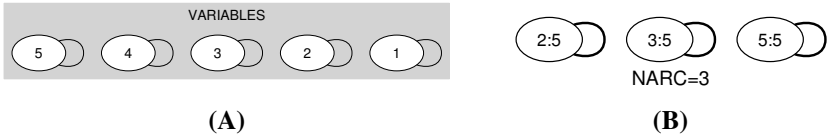


Figure 4.109: Initial and final graph of the count constraint

Automaton Figure 4.110 depicts the automaton associated to the count constraint. To each variable $\texttt{VAR}_i$ of the collection VARIABLES corresponds a 0-1 signature variable $\texttt{S}_i$. The following signature constraint links $\texttt{VAR}_i$ and $\texttt{S}_i$: $\texttt{VAR}_i = \texttt{VALUE} \Leftrightarrow \texttt{S}_i$.

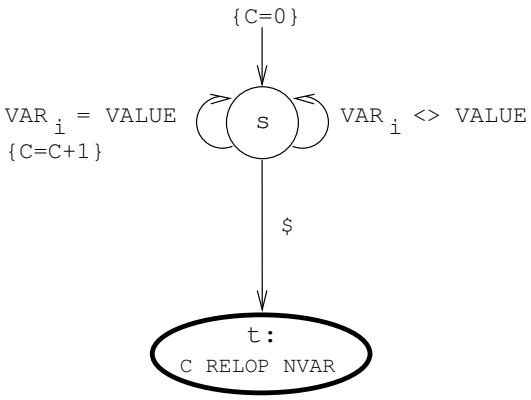


Figure 4.110: Automaton of the `count` constraint

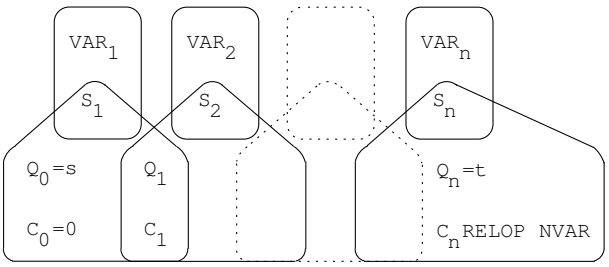


Figure 4.111: Hypergraph of the reformulation corresponding to the automaton of the `count` constraint

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 $\in [=, \neq, <, \geq, >, \leq]$

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

$$\texttt{counts}\left(\begin{array}{l}\{\texttt{val}-1, \texttt{val}-3, \texttt{val}-4, \texttt{val}-9\},\\ \left\{\begin{array}{l}\texttt{var}-4,\\ \texttt{var}-5,\\ \texttt{var}-5,\\ \texttt{var}-4,\\ \texttt{var}-1,\\ \texttt{var}-5\end{array}\right\}, =, 3\end{array}\right)$$

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.

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 $\texttt{VAR}_i$ of the collection VARIABLES corresponds a 0-1 signature variable $\texttt{S}_i$. The following signature constraint links $\texttt{VAR}_i$ and $\texttt{S}_i$: $\texttt{VAR}_i \in \texttt{VALUES} \Leftrightarrow \texttt{S}_i$.

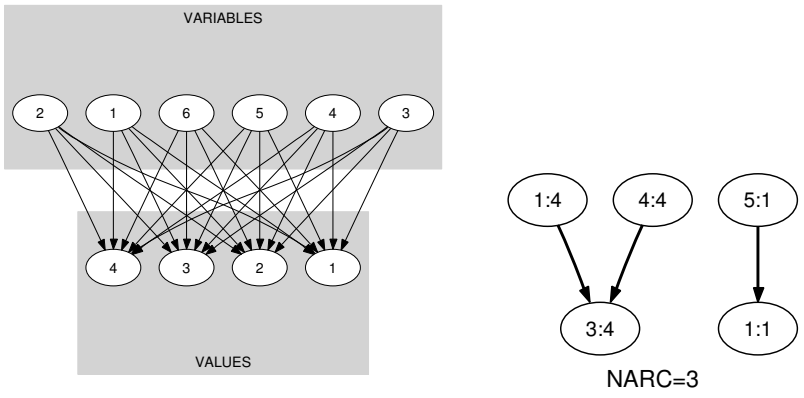


Figure 4.112: Initial and final graph of the `counts` constraint

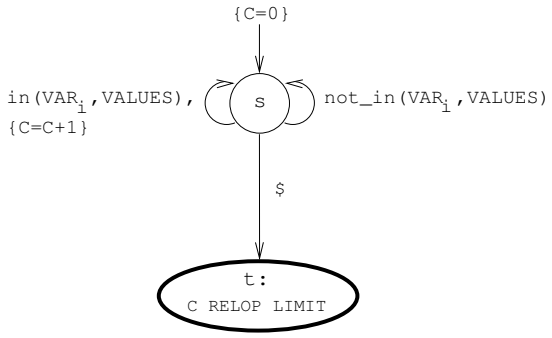


Figure 4.113: Automaton of the `counts` constraint

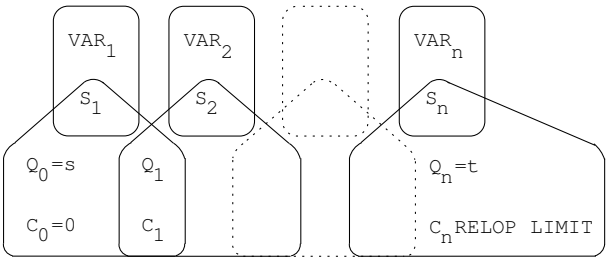


Figure 4.114: Hypergraph of the reformulation corresponding to the automaton of the `counts` constraint

Usage Used in the **Constraint(s) on sets** slot for defining some constraints like `assign_and_counts`.

Used in `assign_and_counts`.

See also `count`, `among`.

Key words value constraint, counting constraint, automaton, automaton with counters, alpha-acyclic constraint network(2), acyclic, bipartite, no_loop.

4.53 crossing

Origin Inspired by [84].

Constraint crossing(NCROSS, SEGMENTS)

Argument(s)

NCROSS	:	dvar
SEGMENTS	:	collection(ox − dvar, oy − dvar, ex − dvar, ey − dvar)

Restriction(s)
- NCROSS ≥ 0
- NCROSS $\leq (|\text{SEGMENTS}| * |\text{SEGMENTS}| - |\text{SEGMENTS}|)/2$
- required(SEGMENTS, [ox, oy, ex, ey])

Purpose NCROSS is the number of line-segments intersections between the line-segments defined by the SEGMENTS collection. Each line-segment is defined by the coordinates (ox, oy) and (ex, ey) of its two extremities.

Arc input(s) SEGMENTS

Arc generator $CLIQUE(<) \mapsto$ collection(s1, s2)

Arc arity 2

Arc constraint(s)
- 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)
- $$\bigvee \left(\begin{array}{l} (\text{s2.ox} - \text{s1.ex}) * (\text{s1.ey} - \text{s1.oy}) - (\text{s1.ex} - \text{s1.ox}) * (\text{s2.oy} - \text{s1.ey}) = 0, \\ (\text{s2.ex} - \text{s1.ex}) * (\text{s2.oy} - \text{s1.oy}) - (\text{s2.ox} - \text{s1.ox}) * (\text{s2.ey} - \text{s1.ey}) = 0, \\ \text{sign}((\text{s2.ox} - \text{s1.ex}) * (\text{s1.ey} - \text{s1.oy}) - (\text{s1.ex} - \text{s1.ox}) * (\text{s2.oy} - \text{s1.ey})) \neq \\ \text{sign}((\text{s2.ex} - \text{s1.ex}) * (\text{s2.oy} - \text{s1.oy}) - (\text{s2.ox} - \text{s1.ox}) * (\text{s2.ey} - \text{s1.ey})) \end{array} \right)$$

Graph property(ies) $\mathbf{NARC} = $ NCROSS

Example
$$\text{crossing}\left(3, \left\{ \begin{array}{llll} \text{ox} - 1 & \text{oy} - 4 & \text{ex} - 9 & \text{ey} - 2, \\ \text{ox} - 1 & \text{oy} - 1 & \text{ex} - 3 & \text{ey} - 5, \\ \text{ox} - 3 & \text{oy} - 2 & \text{ex} - 7 & \text{ey} - 4, \\ \text{ox} - 9 & \text{oy} - 1 & \text{ex} - 9 & \text{ey} - 4 \end{array} \right\} \right)$$

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.

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 $<$ 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.

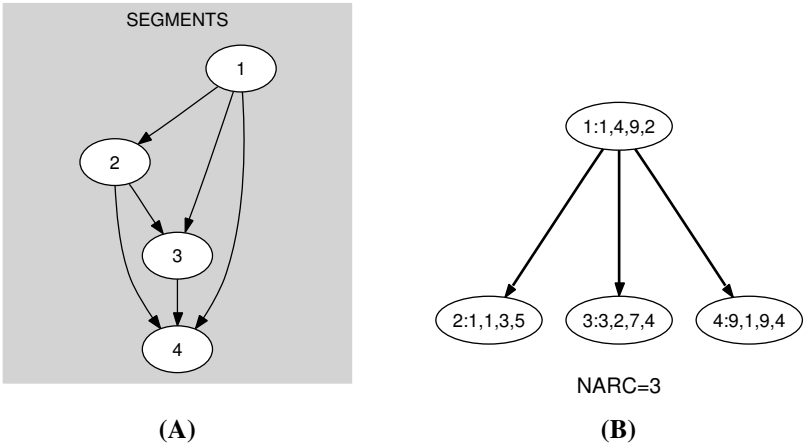


Figure 4.115: Initial and final graph of the `crossing` constraint

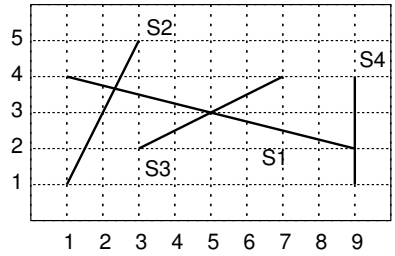


Figure 4.116: Intersection between line-segments

See also `graph_crossing`, `two_layer_edge_crossing`.

Key words geometrical constraint, line-segments intersection, no_loop.

4.54 cumulative

Origin [67]

Constraint cumulative(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 ≥ 0
TASKS.height ≥ 0
LIMIT ≥ 0

Purpose

Cumulative scheduling constraint or scheduling under resource constraints. Consider a set $\mathcal{T}$ of tasks described by the TASKS collection. The cumulative 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 origin + duration = end.

Arc input(s) TASKS

Arc generator $SELF \mapsto$ collection(tasks)

Arc arity 1

Arc constraint(s) tasks.origin + tasks.duration = tasks.end

Graph property(ies) $\mathbf{NARC} = |$TASKS$|$

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 $<$ tasks2.end

Sets

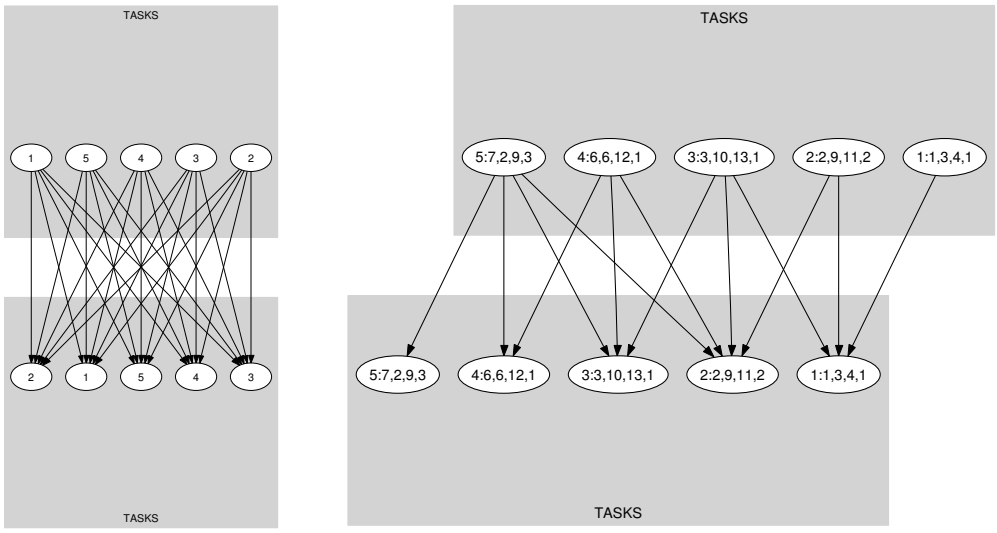
$$\text{SUCC} \mapsto \left[\begin{array}{l} \text{source}, \\ \text{variables} - \text{col}\left(\begin{array}{l} \text{VARIABLES} - \text{collection}(\text{var} - \text{dvar}), \\ [\text{item}(\text{var} - \text{TASKS.height})] \end{array}\right) \end{array}\right]$$

Constraint(s) on sets sum_ctr(variables, $\leq$, LIMIT)

Example

$$\texttt{cumulative}\left(\left\{\begin{array}{llll} \texttt{origin}-1 & \texttt{duration}-3 & \texttt{end}-4 & \texttt{height}-1, \\ \texttt{origin}-2 & \texttt{duration}-9 & \texttt{end}-11 & \texttt{height}-2, \\ \texttt{origin}-3 & \texttt{duration}-10 & \texttt{end}-13 & \texttt{height}-1, \\ \texttt{origin}-6 & \texttt{duration}-6 & \texttt{end}-12 & \texttt{height}-1, \\ \texttt{origin}-7 & \texttt{duration}-2 & \texttt{end}-9 & \texttt{height}-3 \end{array}\right\}, 8\right)$$

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 $\mathcal{S}$ of the final graph the sum of the heights of the tasks in $\mathcal{S}$ does not exceed the limit $\texttt{LIMIT} = 8$. Figure 4.118 shows the cumulated profile associated to the previous example.



(A) **(B)**

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 $\mathbf{NARC} = |\texttt{TASKS}|$ to $\mathbf{NARC} \geq |\texttt{TASKS}|$. This leads to simplify $\underline{\mathbf{NARC}}$ to $\overline{\mathbf{NARC}}$.

Automaton Figure 4.119 depicts the automaton associated to the `cumulative` constraint. To each item of the collection `TASKS` corresponds a signature variable $\texttt{S}_i$, which is equal to 1.

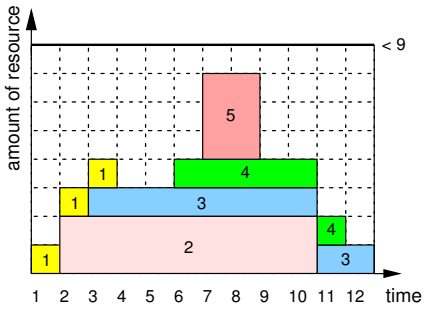


Figure 4.118: Resource consumption profile

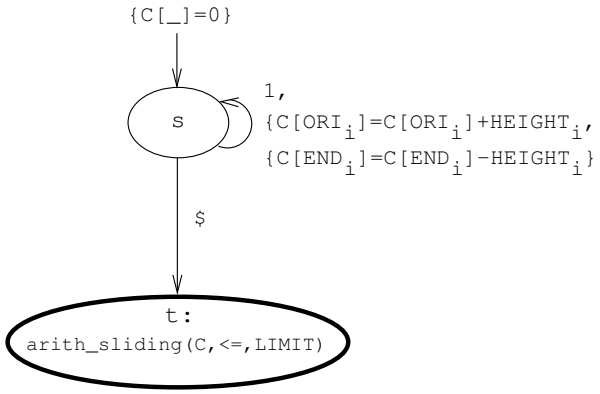


Figure 4.119: Automaton of the `cumulative` constraint

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 ≥ 0
TASKS.height ≥ 1
LIMIT ≥ 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.

Arc input(s) TASKS

Arc generator $SELF \mapsto$ collection(tasks)

Arc arity 1

Arc constraint(s) tasks.origin + tasks.duration = tasks.end

Graph property(ies) $\mathbf{NARC} = |$TASKS$|$

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 $<$ tasks2.end

Sets

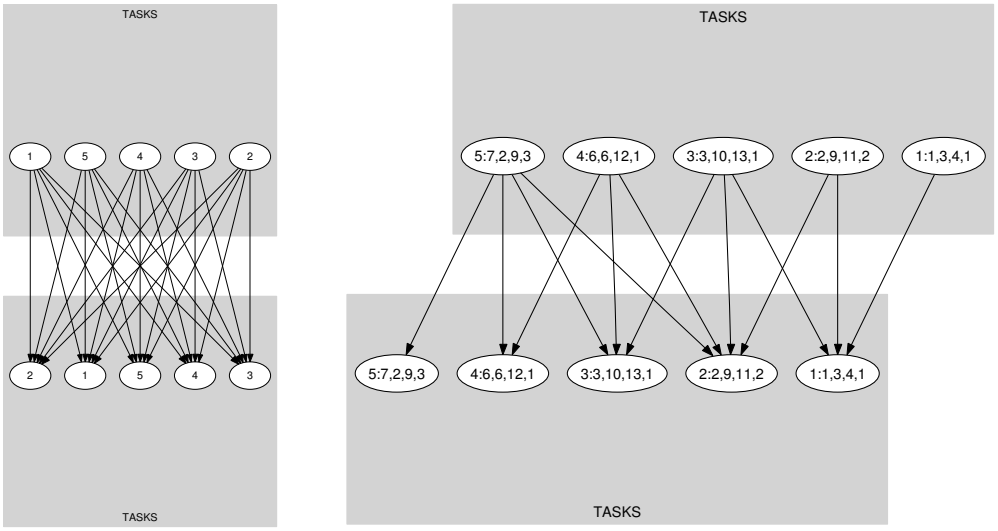
$$\text{SUCC} \mapsto \left[\begin{array}{l} \text{source}, \\ \text{variables} - \text{col}\left(\begin{array}{l} \text{VARIABLES} - \text{collection}(\text{var} - \text{dvar}), \\ [\text{item}(\text{var} - \text{ITEMS.height})] \end{array}\right) \end{array}\right]$$

Constraint(s) on sets product_ctr(variables, $\leq$, LIMIT)

Example

$$\texttt{cumulative\_product}\left(\left\{\begin{array}{llll} \texttt{origin}-1 & \texttt{duration}-3 & \texttt{end}-4 & \texttt{height}-1, \\ \texttt{origin}-2 & \texttt{duration}-9 & \texttt{end}-11 & \texttt{height}-2, \\ \texttt{origin}-3 & \texttt{duration}-10 & \texttt{end}-13 & \texttt{height}-1, \\ \texttt{origin}-6 & \texttt{duration}-6 & \texttt{end}-12 & \texttt{height}-1, \\ \texttt{origin}-7 & \texttt{duration}-2 & \texttt{end}-9 & \texttt{height}-3 \end{array}\right\}, 6\right)$$

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 $\texttt{LIMIT} = 6$. Figure 4.121 shows the solution associated to the previous example.



(A) **(B)**

Figure 4.120: Initial and final graph of the `cumulative_product` constraint

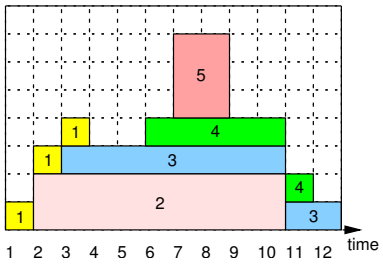


Figure 4.121: Solution of the `cumulative_product` constraint

Signature Since TASKS is the maximum number of vertices of the final graph of the first graph constraint we can rewrite $\mathbf{NARC} = |\texttt{TASKS}|$ to $\mathbf{NARC} \geq |\texttt{TASKS}|$. This leads to simplify $\underline{\overline{\mathbf{NARC}}}$ to $\overline{\mathbf{NARC}}$.

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	$\left(\begin{array}{l}\texttt{start1}-\texttt{dvar},\\ \texttt{size1}-\texttt{dvar},\\ \texttt{last1}-\texttt{dvar},\\ \texttt{start2}-\texttt{dvar},\\ \texttt{size2}-\texttt{dvar},\\ \texttt{last2}-\texttt{dvar},\\ \texttt{height}-\texttt{dvar}\end{array}\right)$
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)

$$\texttt{col}\left(\begin{array}{l}\texttt{CORNERS}-\texttt{collection}(\texttt{size1}-\texttt{dvar},\texttt{size2}-\texttt{dvar},\texttt{x}-\texttt{dvar},\texttt{y}-\texttt{dvar}),\\ \left[\begin{array}{l}\texttt{item}\left(\begin{array}{l}\texttt{size1}-\texttt{RECTANGLES.size1},\\ \texttt{size2}-\texttt{RECTANGLES.size2},\\ \texttt{x}-\texttt{RECTANGLES.start1},\\ \texttt{y}-\texttt{RECTANGLES.start2}\end{array}\right),\\ \texttt{item}\left(\begin{array}{l}\texttt{size1}-\texttt{RECTANGLES.size1},\\ \texttt{size2}-\texttt{RECTANGLES.size2},\\ \texttt{x}-\texttt{RECTANGLES.start1},\\ \texttt{y}-\texttt{RECTANGLES.last2}\end{array}\right),\\ \texttt{item}\left(\begin{array}{l}\texttt{size1}-\texttt{RECTANGLES.size1},\\ \texttt{size2}-\texttt{RECTANGLES.size2},\\ \texttt{x}-\texttt{RECTANGLES.last1},\\ \texttt{y}-\texttt{RECTANGLES.start2}\end{array}\right),\\ \texttt{item}\left(\begin{array}{l}\texttt{size1}-\texttt{RECTANGLES.size1},\\ \texttt{size2}-\texttt{RECTANGLES.size2},\\ \texttt{x}-\texttt{RECTANGLES.last1},\\ \texttt{y}-\texttt{RECTANGLES.last2}\end{array}\right)\end{array}\right]\end{array}\right)$$

Arc input(s) RECTANGLES

Arc generator $SELF \mapsto$ collection(rectangles)

Arc arity 1

Arc constraint(s)
- rectangles.start1 + rectangles.size1 − 1 = rectangles.last1
- rectangles.start2 + rectangles.size2 − 1 = rectangles.last2

Graph property(ies) $\mathbf{NARC} = |\texttt{RECTANGLES}|$

Arc input(s) CORNERS RECTANGLES

Arc generator $PRODUCT \mapsto \texttt{collection}(\texttt{corners}, \texttt{rectangles})$

Arc arity 2

Arc constraint(s)
- corners.size1 > 0
- corners.size2 > 0
- rectangles.start1 ≤ corners.x
- corners.x ≤ rectangles.last1
- rectangles.start2 ≤ corners.y
- corners.y ≤ rectangles.last2

Sets
$$\text{SUCC} \mapsto \left[\begin{array}{l}\texttt{source},\\ \texttt{variables} - \texttt{col}\left(\begin{array}{l}\texttt{VARIABLES} - \texttt{collection}(\texttt{var} - \texttt{dvar}),\\ [\texttt{item}(\texttt{var} - \texttt{RECTANGLES.height})]\end{array}\right)\end{array}\right]$$

Constraint(s) on sets sum_ctr(variables, ≤, LIMIT)

Example
$$\texttt{cumulative\_two\_d}\left(\left\{\begin{array}{lllllll}\texttt{start1}-1 & \texttt{size1}-4 & \texttt{last1}-4 & \texttt{start2}-3 & \texttt{size2}-3 & \texttt{last2}-5 & \texttt{height}-4,\\ \texttt{start1}-3 & \texttt{size1}-2 & \texttt{last1}-4 & \texttt{start2}-1 & \texttt{size2}-2 & \texttt{last2}-2 & \texttt{height}-2,\\ \texttt{start1}-1 & \texttt{size1}-2 & \texttt{last1}-2 & \texttt{start2}-1 & \texttt{size2}-2 & \texttt{last2}-2 & \texttt{height}-3,\\ \texttt{start1}-4 & \texttt{size1}-1 & \texttt{last1}-4 & \texttt{start2}-1 & \texttt{size2}-1 & \texttt{last2}-1 & \texttt{height}-1\end{array}\right\}, 4\right)$$

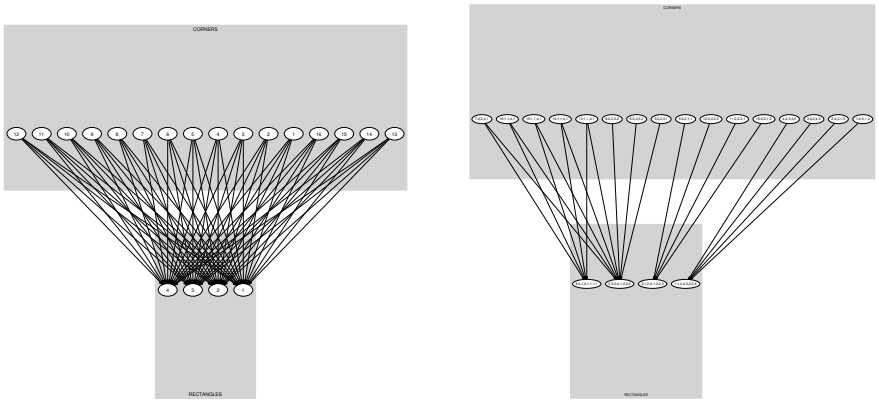
Parts (A) and (B) of Figure 4.122 respectively show the initial and final graph associated to the second graph constraint. On the one hand, each source vertex of the final graph corresponds to the corner of a rectangle of the RECTANGLES collection. On the other hand the successors of a source vertex are those rectangles which overlap that corner.

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.

Signature Since RECTANGLES is the maximum number of vertices of the final graph of the first graph constraint we can rewrite $\mathbf{NARC} = |\texttt{RECTANGLES}|$ to $\mathbf{NARC} \geq |\texttt{RECTANGLES}|$. This leads to simplify $\underline{\mathbf{NARC}}$ to $\overline{\mathbf{NARC}}$.

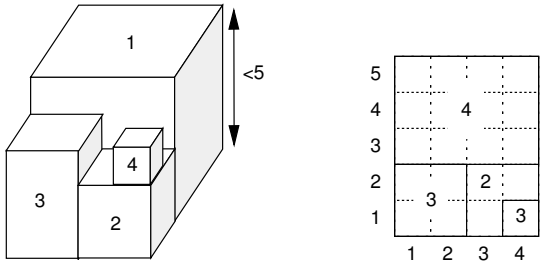
Usage The cumulative_two_d constraint is a necessary condition for the diffn constraint in 3 dimensions (i.e. the placement of parallelepipeds in such a way that they do not pairwise overlap and that each parallelepiped has his sides parallel to the sides of the placement space).

Algorithm A first natural way to handle this constraint would be to accumulate the *compulsory parts* [85] of the rectangles in a quadtree [88]. To each leave of the quadtree we associate the cumulated height of the rectangles containing the corresponding region.



(A) **(B)**

Figure 4.122: Initial and final graph of the `cumulative_two_d` constraint



(A) **(B)**

Figure 4.123: Two representations of a 2-dimensional cumulated profile

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)

TASKS	:	collection	$\left(\begin{array}{l}\texttt{priority} - \texttt{int},\\ \texttt{origin} - \texttt{dvar},\\ \texttt{duration} - \texttt{dvar},\\ \texttt{end} - \texttt{dvar},\\ \texttt{height} - \texttt{dvar}\end{array}\right)$
PRIORITIES	:	collection(id − int, capacity − int)	

Restriction(s)

required(TASKS, [priority, height])
require_at_least(2, TASKS, [origin, duration, end])
TASKS.priority ≥ 1
TASKS.priority $\leq$ |PRIORITIES|
TASKS.duration ≥ 0
TASKS.height ≥ 0
required(PRIORITIES, [id, capacity])
PRIORITIES.id ≥ 1
PRIORITIES.id $\leq$ |PRIORITIES|
increasing_seq(PRIORITIES, id)
increasing_seq(PRIORITIES, capacity)

Purpose

Consider a set $\mathcal{T}$ of tasks described by the TASKS collection where each task has a given priority choosen in the range $[1, \texttt{PRIORITIES}]$. 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 origin + duration = end.

Derived Collection(s)

$$\texttt{col}\left(\begin{array}{l}\texttt{TIME\_POINTS} - \texttt{collection}(\texttt{idp} - \texttt{int}, \texttt{duration} - \texttt{dvar}, \texttt{point} - \texttt{dvar}),\\ \left[\begin{array}{l}\texttt{item}(\texttt{idp} - \texttt{TASKS.priority}, \texttt{duration} - \texttt{TASKS.duration}, \texttt{point} - \texttt{TASKS.origin}),\\ \texttt{item}(\texttt{idp} - \texttt{TASKS.priority}, \texttt{duration} - \texttt{TASKS.duration}, \texttt{point} - \texttt{TASKS.end})\end{array}\right]\end{array}\right)$$

Arc input(s) TASKS

Arc generator $SELF \mapsto$ collection(tasks)

Arc arity 1

Arc constraint(s) tasks.origin + tasks.duration = tasks.end

Graph property(ies) $\mathbf{NARC} =$ |TASKS|

For all items of PRIORITIES:

Arc input(s) TIME_POINTS TASKS

Arc generator $PRODUCT \mapsto \texttt{collection}(\texttt{time\_points}, \texttt{tasks})$

Arc arity 2

Arc constraint(s)
- $\texttt{time\_points.idp} = \texttt{PRIORITIES.id}$
- $\texttt{time\_points.idp} \geq \texttt{tasks.priority}$
- $\texttt{time\_points.duration} > 0$
- $\texttt{tasks.origin} \leq \texttt{time\_points.point}$
- $\texttt{time\_points.point} < \texttt{tasks.end}$

Sets
$$\begin{array}{l}\mathsf{SUCC} \mapsto \\ \left[\begin{array}{l}\texttt{source}, \\ \texttt{variables} - \texttt{col}\left(\begin{array}{l}\texttt{VARIABLES} - \texttt{collection}(\texttt{var} - \texttt{dvar}), \\ [\texttt{item}(\texttt{var} - \texttt{TASKS.height})]\end{array}\right)\end{array}\right]\end{array}$$

Constraint(s) on sets $\texttt{sum\_ctr}(\texttt{variables}, \leq, \texttt{PRIORITIES.capacity})$

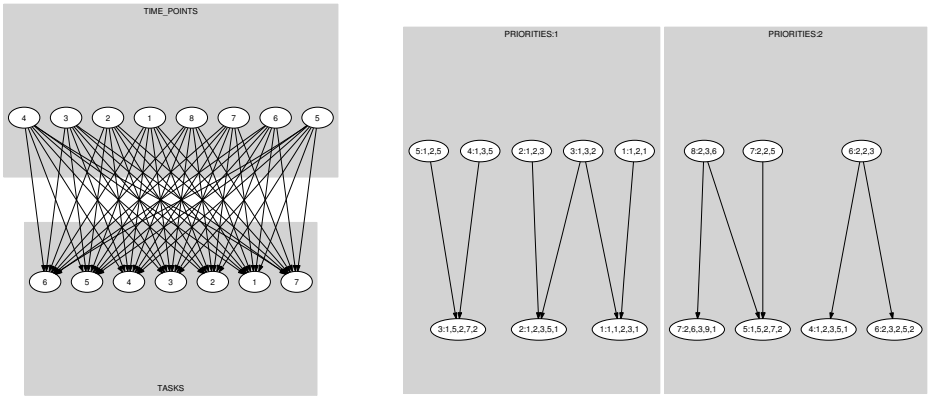
Example
$$\texttt{cumulative\_with\_level\_of\_priority}\left(\begin{array}{l}\left\{\begin{array}{lllll}\texttt{priority}-1 & \texttt{origin}-1 & \texttt{duration}-2 & \texttt{end}-3 & \texttt{height}-1, \\ \texttt{priority}-1 & \texttt{origin}-2 & \texttt{duration}-3 & \texttt{end}-5 & \texttt{height}-1, \\ \texttt{priority}-1 & \texttt{origin}-5 & \texttt{duration}-2 & \texttt{end}-7 & \texttt{height}-2, \\ \texttt{priority}-2 & \texttt{origin}-3 & \texttt{duration}-2 & \texttt{end}-5 & \texttt{height}-2, \\ \texttt{priority}-2 & \texttt{origin}-6 & \texttt{duration}-3 & \texttt{end}-9 & \texttt{height}-1\end{array}\right\}, \\ \left\{\begin{array}{ll}\texttt{id}-1 & \texttt{capacity}-2, \\ \texttt{id}-2 & \texttt{capacity}-3\end{array}\right\}\end{array}\right)$$

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 $\mathbf{NARC} = |\texttt{TASKS}|$ to $\mathbf{NARC} \geq |\texttt{TASKS}|$. This leads to simplify $\underline{\mathbf{NARC}}$ to $\overline{\mathbf{NARC}}$.

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 $1, 2, \ldots, i$ 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



(A) **(B)**

Figure 4.124: Initial and final graph of the `cumulative_with_level_of_priority` constraint

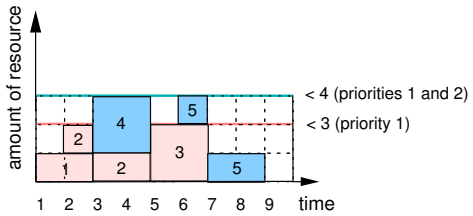


Figure 4.125: Resource consumption profile according to both levels of priority

$$\texttt{cumulative\_with\_level\_of\_priority}\left(\begin{array}{l}\left\{\begin{array}{llll}\texttt{priority}-1 & \texttt{origin}-o_1 & \texttt{duration}-2 & \texttt{height}-2,\\ \texttt{priority}-1 & \texttt{origin}-o_2 & \texttt{duration}-2 & \texttt{height}-1,\\ \texttt{priority}-2 & \texttt{origin}-o_3 & \texttt{duration}-1 & \texttt{height}-3\end{array}\right\},\\ \left\{\begin{array}{ll}\texttt{id}-1 & \texttt{capacity}-2,\\ \texttt{id}-2 & \texttt{capacity}-3\end{array}\right\}\end{array}\right)$$

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

$$\texttt{cumulative}\left(\left\{\begin{array}{lll}\texttt{origin}-o_1 & \texttt{duration}-2 & \texttt{height}-2,\\ \texttt{origin}-o_2 & \texttt{duration}-2 & \texttt{height}-1\end{array}\right\},2\right)$$

$$\texttt{cumulative}\left(\left\{\begin{array}{lll}\texttt{origin}-o_1 & \texttt{duration}-2 & \texttt{height}-2,\\ \texttt{origin}-o_2 & \texttt{duration}-2 & \texttt{height}-1,\\ \texttt{origin}-o_3 & \texttt{duration}-1 & \texttt{height}-3\end{array}\right\},3\right)$$

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 cumulatives(TASKS, MACHINES, CTR)

Argument(s)

TASKS	:	collection(machine − dvar, origin − dvar, duration − dvar, end − dvar, height − dvar)
MACHINES	:	collection(id − int, capacity − int)
CTR	:	atom

Restriction(s)

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 $\in [\leq, \geq]$

Purpose

Consider a set $\mathcal{T}$ of tasks described by the TASKS collection. When CTR is equal to $\leq$ (repectively $\geq$), the cumulatives 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 origin + duration = end.

Derived Collection(s)

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)])

Arc input(s) TASKS

Arc generator $SELF \mapsto$ collection(tasks)

Arc arity 1

Arc constraint(s) tasks.origin + tasks.duration = tasks.end

Graph property(ies) $\mathbf{NARC} = |$TASKS$|$

For all items of MACHINES:

Arc input(s) TIME_POINTS TASKS

Arc generator $PRODUCT \mapsto$ collection(time_points, tasks)

Arc arity 2

Arc constraint(s)
- time_points.idm = MACHINES.id
- time_points.idm = tasks.machine
- time_points.duration > 0
- tasks.origin $\leq$ time_points.point
- time_points.point $<$ tasks.end

Sets
$$\text{SUCC} \mapsto \left[\begin{array}{l} \texttt{source}, \\ \texttt{variables} - \texttt{col}\left(\begin{array}{l} \texttt{VARIABLES} - \texttt{collection}(\texttt{var} - \texttt{dvar}), \\ [\texttt{item}(\texttt{var} - \texttt{TASKS.height})] \end{array} \right) \end{array} \right]$$

Constraint(s) on sets sum_ctr(variables, CTR, MACHINES.capacity)

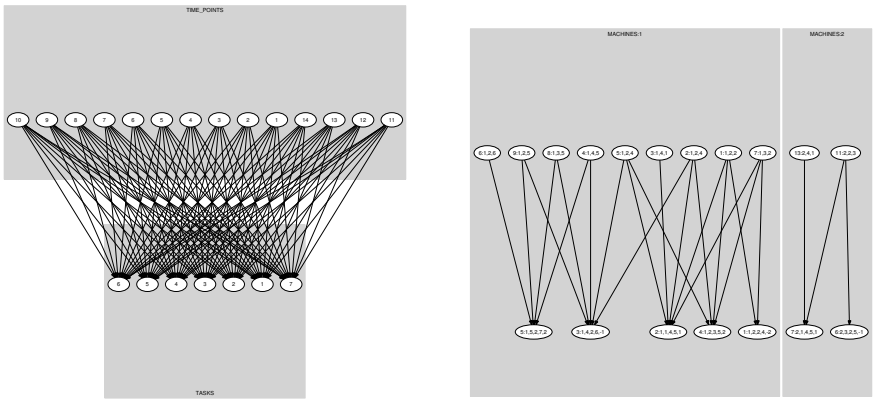
Example
$$\texttt{cumulatives}\left(\begin{array}{l} \left\{ \begin{array}{lllll} \texttt{machine}-1 & \texttt{origin}-2 & \texttt{duration}-2 & \texttt{end}-4 & \texttt{height}--2, \\ \texttt{machine}-1 & \texttt{origin}-1 & \texttt{duration}-4 & \texttt{end}-5 & \texttt{height}-1, \\ \texttt{machine}-1 & \texttt{origin}-4 & \texttt{duration}-2 & \texttt{end}-6 & \texttt{height}--1, \\ \texttt{machine}-1 & \texttt{origin}-2 & \texttt{duration}-3 & \texttt{end}-5 & \texttt{height}-2, \\ \texttt{machine}-1 & \texttt{origin}-5 & \texttt{duration}-2 & \texttt{end}-7 & \texttt{height}-2, \\ \texttt{machine}-2 & \texttt{origin}-3 & \texttt{duration}-2 & \texttt{end}-5 & \texttt{height}--1, \\ \texttt{machine}-2 & \texttt{origin}-1 & \texttt{duration}-4 & \texttt{end}-5 & \texttt{height}-1 \end{array} \right\}, \\ \{\texttt{id}-1\ \texttt{capacity}-0, \texttt{id}-2\ \texttt{capacity}-0\}, \geq \end{array} \right)$$

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 greather 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 $\mathbf{NARC} = |\texttt{TASKS}|$ to $\mathbf{NARC} \geq |\texttt{TASKS}|$. This leads to simplify $\underline{\overline{\mathbf{NARC}}}$ to $\overline{\mathbf{NARC}}$.

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



(A) **(B)**

Figure 4.126: Initial and final graph of the `cumulatives` constraint

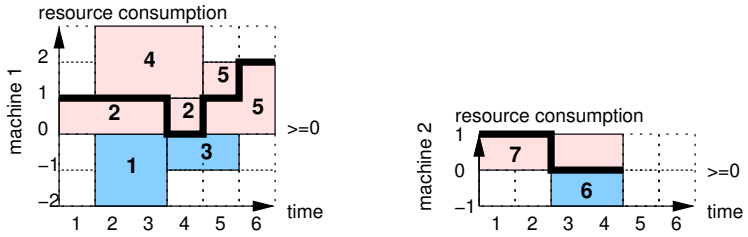


Figure 4.127: Resource consumption profile on the different machines

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 $\leq |$NODES$|$
required(NODES, [index, succ, bool])
NODES.index ≥ 1
NODES.index $\leq |$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 $n -$ SIZE_CUTSET and their corresponding arcs form a graph without circuit.

Arc input(s) NODES

Arc generator $CLIQUE \mapsto$ collection(nodes1, nodes2)

Arc arity 2

Arc constraint(s)

- in_set(nodes2.index, nodes1.succ)
- nodes1.bool $= 1$
- nodes2.bool $= 1$

Graph property(ies)

- **MAX_NSCC** ≤ 1
- **NVERTEX** $=$ |NODES| $-$ SIZE_CUTSET

Example

$$\texttt{cutset}\left(\begin{array}{l} 1, \left\{\begin{array}{lll} \texttt{index} - 1 & \texttt{succ} - \{2,3,4\} & \texttt{bool} - 1, \\ \texttt{index} - 2 & \texttt{succ} - \{3\} & \texttt{bool} - 1, \\ \texttt{index} - 3 & \texttt{succ} - \{4\} & \texttt{bool} - 1, \\ \texttt{index} - 4 & \texttt{succ} - \{1\} & \texttt{bool} - 0 \end{array}\right\} \end{array}\right)$$

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 $\mathbf{MAX\_NSCC} \leq 1$ 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.

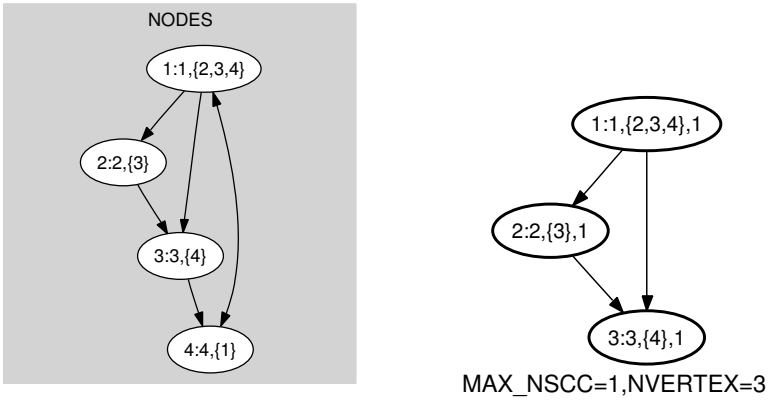


Figure 4.128: Initial and final graph of the `cutset` set constraint

4.60 cycle

Origin [37]

Constraint cycle(NCYCLE, NODES)

Argument(s)

NCYCLE	:	dvar
NODES	:	collection(index − int, succ − dvar)

Restriction(s)

NCYCLE ≥ 1
NCYCLE $\leq$ |NODES|
required(NODES, [index, succ])
NODES.index ≥ 1
NODES.index $\leq$ |NODES|
distinct(NODES, index)
NODES.succ ≥ 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) NODES

Arc generator $CLIQUE \mapsto$ collection(nodes1, nodes2)

Arc arity 2

Arc constraint(s) nodes1.succ = nodes2.index

Graph property(ies)
- $\mathbf{NTREE} = 0$
- $\mathbf{NCC} =$ NCYCLE

Example

$$\texttt{cycle}\left(\begin{array}{l} 2, \left\{\begin{array}{ll} \texttt{index} - 1 & \texttt{succ} - 2, \\ \texttt{index} - 2 & \texttt{succ} - 1, \\ \texttt{index} - 3 & \texttt{succ} - 5, \\ \texttt{index} - 4 & \texttt{succ} - 3, \\ \texttt{index} - 5 & \texttt{succ} - 4 \end{array}\right\} \end{array}\right)$$

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 $\mathbf{NCC}$ 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. $\mathbf{NTREE} = 0$) and since NCYCLE $= \mathbf{NCC} = 2$.

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 $[1, |\texttt{NODES}|]$. 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 $\mathbf{NTREE} = 0$ 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 their was a constraint named `circuit` which, from a declarative point of view, was equivalent to `cycle`$(1, \texttt{NODES})$. 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 $\max(\texttt{NCYCLE})$ 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.

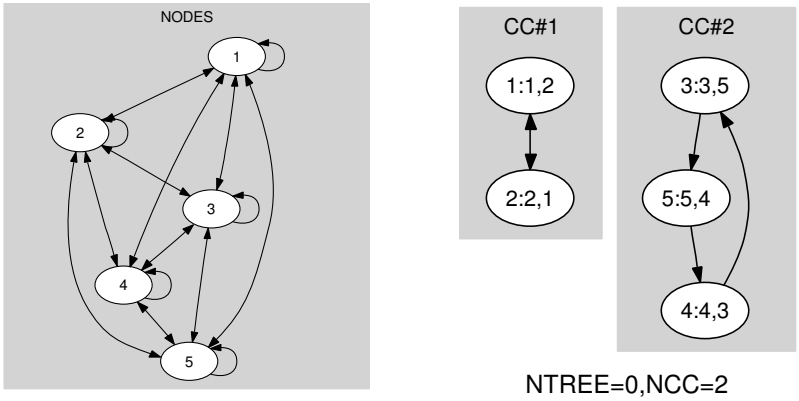


Figure 4.129: Initial and final graph of the `cycle` constraint

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 ≥ 1
NCYCLE $\leq |$NODES$|$
required(NODES, [index, succ, colour])
NODES.index ≥ 1
NODES.index $\leq |$NODES$|$
distinct(NODES, index)
NODES.succ ≥ 1
NODES.succ $\leq |$NODES$|$
ATLEAST ≥ 0
ATLEAST $\leq$ PATH_LEN
ATMOST $\geq$ ATLEAST
PATH_LEN ≥ 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 $CLIQUE \mapsto$ collection(nodes1, nodes2)

Arc arity 2

Arc constraint(s) nodes1.succ = nodes2.index

Graph property(ies)

- $\mathbf{NTREE} = 0$
- $\mathbf{NCC} =$ NCYCLE

Sets

$$\text{PATH\_LENGTH}(\text{PATH\_LEN}) \mapsto \left[\text{variables} - \text{col} \left(\begin{array}{l} \text{VARIABLES} - \text{collection}(\text{var} - \text{dvar}), \\ [\text{item}(\text{var} - \text{NODES.colour})] \end{array} \right) \right]$$

Constraint(s) on sets among_low_up(ATLEAST, ATMOST, variables, VALUES)

Example

$$\texttt{cycle\_card\_on\_path}\left(\begin{array}{l} 2, \left\{\begin{array}{lll} \texttt{index}-1 & \texttt{succ}-7 & \texttt{colour}-2,\\ \texttt{index}-2 & \texttt{succ}-4 & \texttt{colour}-3,\\ \texttt{index}-3 & \texttt{succ}-8 & \texttt{colour}-2,\\ \texttt{index}-4 & \texttt{succ}-9 & \texttt{colour}-1,\\ \texttt{index}-5 & \texttt{succ}-1 & \texttt{colour}-2,\\ \texttt{index}-6 & \texttt{succ}-2 & \texttt{colour}-1,\\ \texttt{index}-7 & \texttt{succ}-5 & \texttt{colour}-1,\\ \texttt{index}-8 & \texttt{succ}-6 & \texttt{colour}-1,\\ \texttt{index}-9 & \texttt{succ}-3 & \texttt{colour}-1 \end{array}\right\}, 1, 2, 3,\\ \{\texttt{val}-1\} \end{array}\right)$$

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. $\mathbf{NTREE} = 0$) and since for each set of three consecutives vertices, colour 1 occurs at least once and at most twice (i.e. the among_low_up constraint holds).

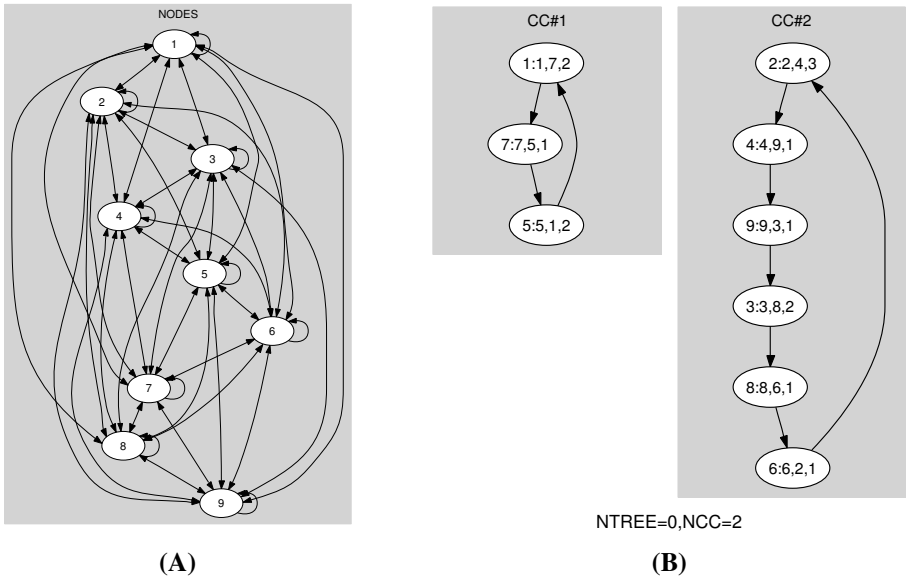


Figure 4.130: Initial and final graph of the cycle_card_on_path constraint

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 consecutives 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.

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 ≥ 0
NCYCLE ≥ 1
NCYCLE $\leq |$NODES$|$
required(NODES, [index, succ, x, y])
NODES.index ≥ 1
NODES.index $\leq |$NODES$|$
distinct(NODES, index)
NODES.succ ≥ 0
NODES.succ $\leq |$NODES$|$
NODES.x ≥ 0
NODES.y ≥ 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)
- $\mathbf{NTREE} = 0$
- $\mathbf{NCC} =$ NCYCLE

Arc input(s) NODES

Arc generator $CLIQUE \mapsto$ collection(nodes1, nodes2)

Arc arity 2

Arc constraint(s)

$$\bigvee \left(\begin{array}{l} \texttt{nodes1.succ} = \texttt{nodes2.index}, \\ \bigwedge \left(\begin{array}{l} \texttt{nodes1.succ} = 0, \\ \texttt{nodes2.succ} \neq 0, \\ \texttt{abs}(\texttt{nodes1.x} - \texttt{nodes2.x}) + \texttt{abs}(\texttt{nodes1.y} - \texttt{nodes2.y}) \leq \texttt{MAXDIST} \end{array} \right) \end{array} \right)$$

Graph property(ies) $\mathbf{NVERTEX} = |\texttt{NODES}|$

Sets
$$\begin{array}{l}\texttt{PRED} \mapsto \\ \left[\begin{array}{l}\texttt{variables} - \texttt{col}(\texttt{VARIABLES} - \texttt{collection}(\texttt{var} - \texttt{dvar}), [\texttt{item}(\texttt{var} - \texttt{NODES.succ})]), \\ \texttt{destination}\end{array}\right]\end{array}$$

Constraint(s) on sets $\texttt{nvalues\_except\_0}(\texttt{variables}, =, 1)$

Example
$$\texttt{cycle\_or\_accessibility}\left(\begin{array}{l} 3, 2, \left\{\begin{array}{llll} \texttt{index}-1 & \texttt{succ}-6 & \texttt{x}-4 & \texttt{y}-5, \\ \texttt{index}-2 & \texttt{succ}-0 & \texttt{x}-9 & \texttt{y}-1, \\ \texttt{index}-3 & \texttt{succ}-0 & \texttt{x}-2 & \texttt{y}-4, \\ \texttt{index}-4 & \texttt{succ}-1 & \texttt{x}-2 & \texttt{y}-6, \\ \texttt{index}-5 & \texttt{succ}-5 & \texttt{x}-7 & \texttt{y}-2, \\ \texttt{index}-6 & \texttt{succ}-4 & \texttt{x}-4 & \texttt{y}-7, \\ \texttt{index}-7 & \texttt{succ}-0 & \texttt{x}-6 & \texttt{y}-4 \end{array}\right\} \end{array}\right)$$

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.

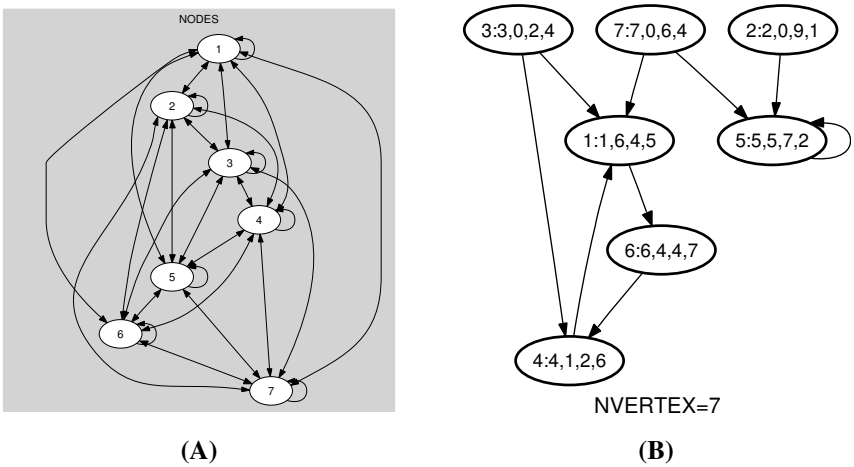


Figure 4.131: Initial and final graph of the `cycle_or_accessibility` constraint

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. nodes1.succ $=$ nodes2.index),
- The arcs between one vertex v_1 that does not belong to any circuit (i.e. nodes1.succ $= 0$) and one vertex v_2 located on a circuit (i.e. 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 $\mathbf{NVERTEX} = |\mathtt{NODES}|$. Finally the dynamic constraint $\mathtt{nvalues\_except\_0}(\mathtt{variables}, =, 1)$ expresses the fact that for each vertex v, there is exactly one predecessor of v which belong to a circuit.

Signature Since |NODES| is the maximum number of vertices of the final graph associated to the second graph constraint we can rewrite $\mathbf{NVERTEX} = |\mathtt{NODES}|$ to $\mathbf{NVERTEX} \geq |\mathtt{NODES}|$. This leads to simplify $\overline{\underline{\mathbf{NVERTEX}}}$ to $\overline{\mathbf{NVERTEX}}$.

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.

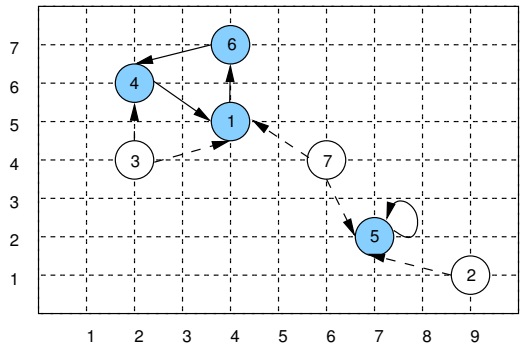


Figure 4.132: Final graph associated to the facilities location problem

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 $t \in$ RESOURCE$[r]$.first_task.
- There is an arc from a task vertex t to a resource vertex r if $r \in$ TASK$[t]$.next_task.
- There is an arc from a task vertex t_1 to a task vertex t_2 if $t_2 \in$ TASK$[t_1]$.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)

$$\text{col}\left(\begin{array}{l}\text{RESOURCE\_TASK} - \text{collection}(\text{index} - \text{int}, \text{succ} - \text{dvar}, \text{name} - \text{dvar}),\\ \left[\begin{array}{l}\text{item}(\text{index} - \text{RESOURCE.id}, \text{succ} - \text{RESOURCE.first\_task}, \text{name} - \text{RESOURCE.id}),\\ \text{item}(\text{index} - \text{TASK.id}, \text{succ} - \text{TASK.next\_task}, \text{name} - \text{TASK.resource})\end{array}\right]\end{array}\right)$$

Arc input(s) RESOURCE_TASK

Arc generator $CLIQUE \mapsto \texttt{collection}(\texttt{resource\_task1}, \texttt{resource\_task2})$

Arc arity 2

Arc constraint(s)
- $\texttt{resource\_task1.succ} = \texttt{resource\_task2.index}$
- $\texttt{resource\_task1.name} = \texttt{resource\_task2.name}$

Graph property(ies)
- $\mathbf{NTREE} = 0$
- $\mathbf{NCC} = |\texttt{RESOURCE}|$
- $\mathbf{NVERTEX} = |\texttt{RESOURCE}| + |\texttt{TASK}|$

For all items of RESOURCE:

Arc input(s) RESOURCE_TASK

Arc generator $CLIQUE \mapsto \texttt{collection}(\texttt{resource\_task1}, \texttt{resource\_task2})$

Arc arity 2

Arc constraint(s)
- $\texttt{resource\_task1.succ} = \texttt{resource\_task2.index}$
- $\texttt{resource\_task1.name} = \texttt{resource\_task2.name}$
- $\texttt{resource\_task1.name} = \texttt{RESOURCE.id}$

Graph property(ies) $\mathbf{NVERTEX} = \texttt{RESOURCE.nb\_task} + 1$

Example
$$\texttt{cycle\_resource}\left(\left\{\begin{array}{llll}\texttt{id}-1 & \texttt{first\_task}-5 & \texttt{nb\_task}-3,\\ \texttt{id}-2 & \texttt{first\_task}-2 & \texttt{nb\_task}-0,\\ \texttt{id}-3 & \texttt{first\_task}-8 & \texttt{nb\_task}-2\end{array}\right\},\left\{\begin{array}{lll}\texttt{id}-4 & \texttt{next\_task}-7 & \texttt{resource}-1,\\ \texttt{id}-5 & \texttt{next\_task}-4 & \texttt{resource}-1,\\ \texttt{id}-6 & \texttt{next\_task}-3 & \texttt{resource}-3,\\ \texttt{id}-7 & \texttt{next\_task}-1 & \texttt{resource}-1,\\ \texttt{id}-8 & \texttt{next\_task}-6 & \texttt{resource}-3\end{array}\right\}\right)$$

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 |RESOURCE| + |TASK| vertices, the corresponding final graph cannot have more than |RESOURCE| + |TASK| vertices. Therefore we can rewrite the graph property $\mathbf{NVERTEX} = |\texttt{RESOURCE}| + |\texttt{TASK}|$ to $\mathbf{NVERTEX} \geq |\texttt{RESOURCE}| + |\texttt{TASK}|$ and simplify $\overline{\mathbf{NVERTEX}}$ to $\overline{\mathbf{NVERTEX}}$.

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:

- The compatibility or incompability 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 `cycle` constraint of CHIP by using the following optional parameters:

- The *resource node* parameter [93, page 97],
- The *circuit weight* parameter [93, page 101],
- The *name* parameter [93, page 104].

See also `cycle`.

Key words graph constraint, resource constraint, graph partitioning constraint, connected component, strongly connected component, derived collection.

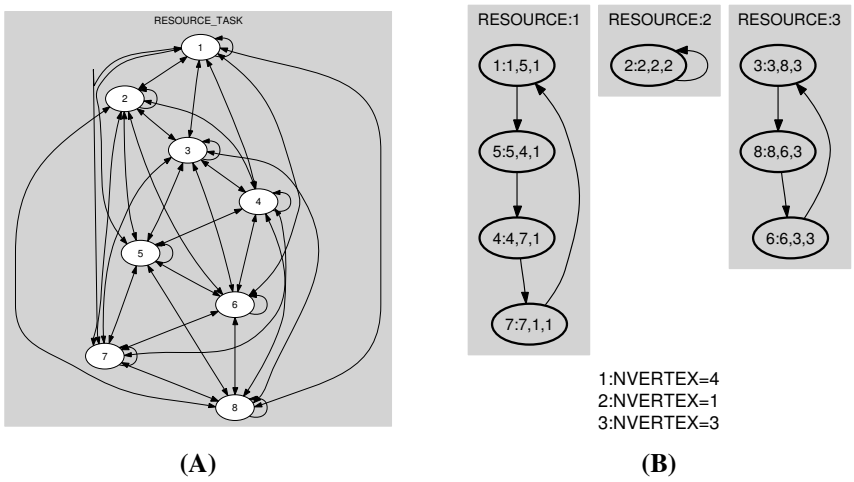


Figure 4.133: Initial and final graph of the `cycle_resource` constraint

4.64 cyclic_change

Origin Derived from change.

Constraint cyclic_change(NCHANGE, CYCLE_LENGTH, VARIABLES, CTR)

Argument(s)

NCHANGE	:	dvar
CYCLE_LENGTH	:	int
VARIABLES	:	collection(var − dvar)
CTR	:	atom

Restriction(s)

NCHANGE ≥ 0
NCHANGE $<$ |VARIABLES|
CYCLE_LENGTH > 0
required(VARIABLES, var)
CTR $\in [=, \neq, <, \geq, >, \leq]$

Purpose

NCHANGE is the number of times that constraint $((X + 1) \bmod$ CYCLE_LENGTH$)$ CTR Y holds; X and Y correspond to consecutive variables of the collection VARIABLES.

Arc input(s) VARIABLES

Arc generator $PATH \mapsto$ collection(variables1, variables2)

Arc arity 2

Arc constraint(s) (variables1.var + 1) mod CYCLE_LENGTH CTR variables2.var

Graph property(ies) **NARC** = NCHANGE

Example 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: $((\text{VAR}_i + 1) \bmod$ CYCLE_LENGTH$)$ CTR $\text{VAR}_{i+1} \Leftrightarrow \text{S}_i$.

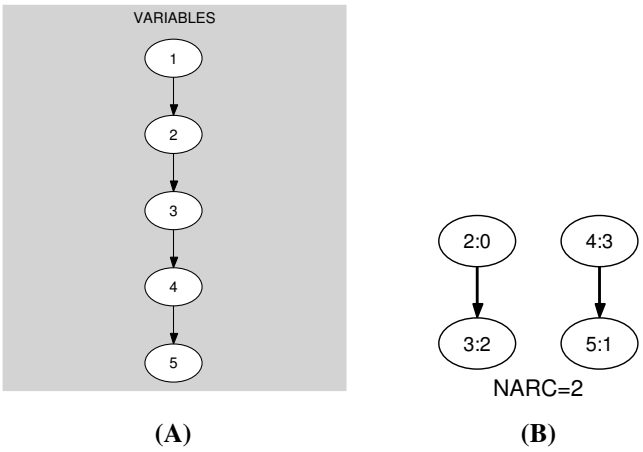


Figure 4.134: Initial and final graph of the `cyclic_change` constraint

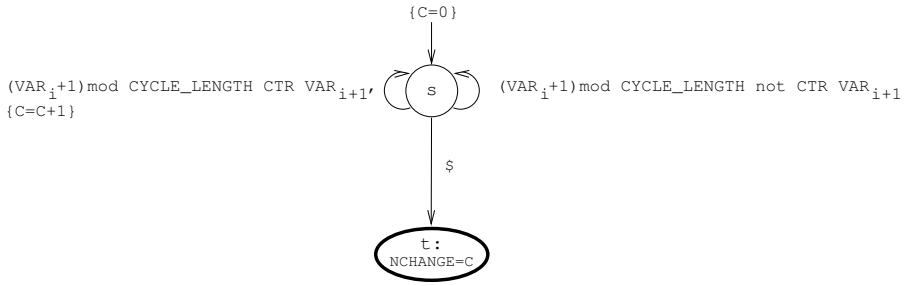


Figure 4.135: Automaton of the `cyclic_change` constraint

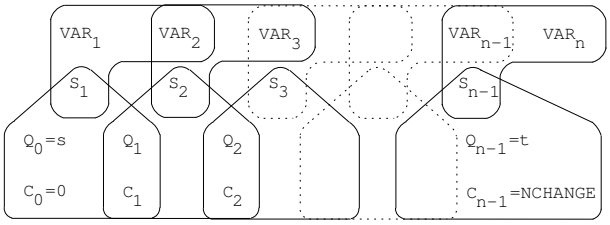


Figure 4.136: Hypergraph of the reformulation corresponding to the automaton of the `cyclic_change` constraint

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.

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 ≥ 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:

$$((X+1) \bmod \texttt{CYCLE\_LENGTH})\ \texttt{CTR}\ Y \wedge X < \texttt{CYCLE\_LENGTH} \wedge Y < \texttt{CYCLE\_LENGTH}$$

X and Y correspond to consecutive variables of the collection VARIABLES.

Arc input(s) VARIABLES

Arc generator $PATH \mapsto$ collection(variables1, variables2)

Arc arity 2

Arc constraint(s)

- $(\texttt{variables1.var} + 1) \bmod \texttt{CYCLE\_LENGTH}\ \texttt{CTR}\ \texttt{variables2.var}$
- variables1.var $<$ CYCLE_LENGTH
- variables2.var $<$ CYCLE_LENGTH

Graph property(ies) $\mathbf{NARC} = \texttt{NCHANGE}$

Example

$$\texttt{cyclic\_change\_joker}\left(\begin{array}{c} 2, 4, \left\{\begin{array}{l} \texttt{var} - 3, \\ \texttt{var} - 0, \\ \texttt{var} - 2, \\ \texttt{var} - 4, \\ \texttt{var} - 4, \\ \texttt{var} - 4, \\ \texttt{var} - 3, \\ \texttt{var} - 1, \\ \texttt{var} - 4 \end{array}\right\}, \neq \end{array}\right)$$

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.

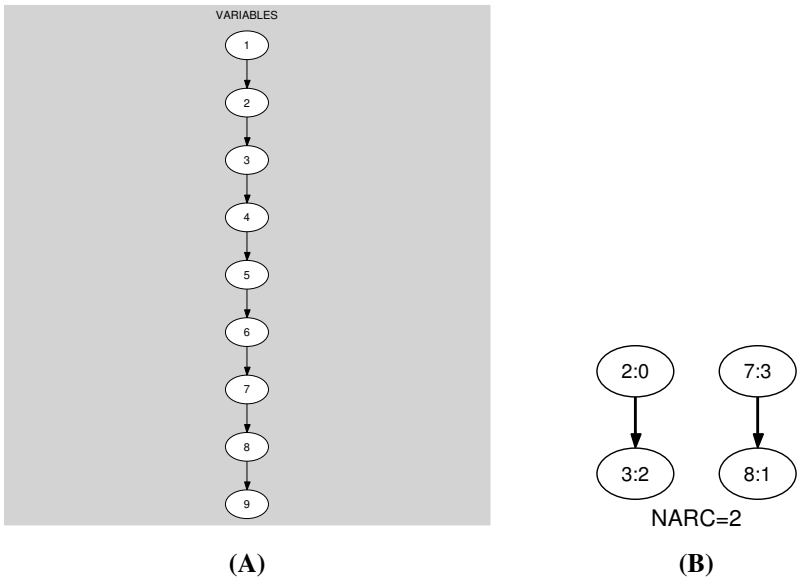


Figure 4.137: Initial and final graph of the `cyclic_change_joker` constraint

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 $(\mathtt{VAR}_i, \mathtt{VAR}_{i+1})$ of the collection VARIABLES corresponds a 0-1 signature variable $\mathtt{S}_i$. The following signature constraint links $\mathtt{VAR}_i$, $\mathtt{VAR}_{i+1}$ and $\mathtt{S}_i$:

$$(((\mathtt{VAR}_i + 1) \bmod \mathtt{CYCLE\_LENGTH})\ \mathtt{CTR}\ \mathtt{VAR}_{i+1} \wedge$$
$$(\mathtt{VAR}_i < \mathtt{CYCLE\_LENGTH}) \wedge (\mathtt{VAR}_{i+1} < \mathtt{CYCLE\_LENGTH})) \Leftrightarrow \mathtt{S}_i.$$

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 holliday. 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 holliday.

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.

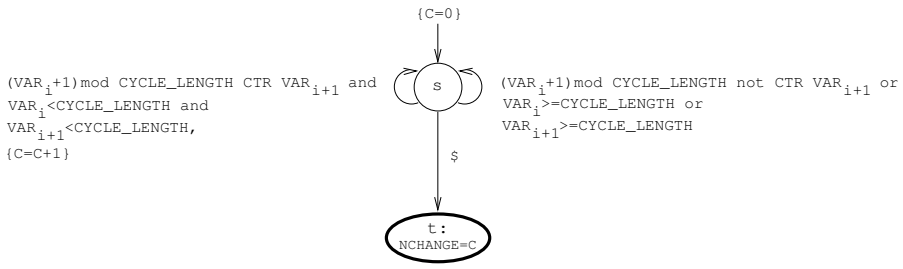


Figure 4.138: Automaton of the `cyclic_change_joker` constraint

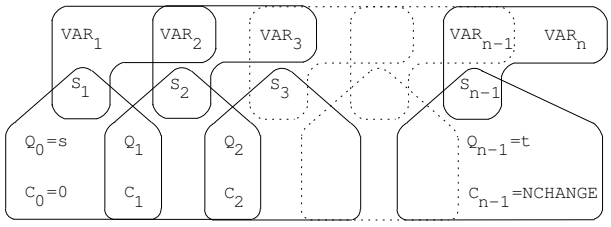


Figure 4.139: Hypergraph of the reformulation corresponding to the automaton of the `cyclic_change_joker` constraint

4.66 decreasing

Origin Inspired by `increasing`.

Constraint `decreasing(VARIABLES)`

Argument(s) `VARIABLES` : `collection(var − dvar)`

Restriction(s) $|\texttt{VARIABLES}| > 0$
`required(VARIABLES, var)`

Purpose The variables of the collection `VARIABLES` are decreasing.

Arc input(s) `VARIABLES`

Arc generator $PATH \mapsto \texttt{collection(variables1, variables2)}$

Arc arity 2

Arc constraint(s) `variables1.var` $\geq$ `variables2.var`

Graph property(ies) $\mathbf{NARC} = |\texttt{VARIABLES}| - 1$

Example `decreasing(`$\{$`var` -8, `var` -4, `var` -1, `var` $-1\}$`)`

Parts (A) and (B) of Figure 4.140 respectively show the initial and final graph. Since we use the $\mathbf{NARC}$ graph property, the arcs of the final graph are stressed in bold.

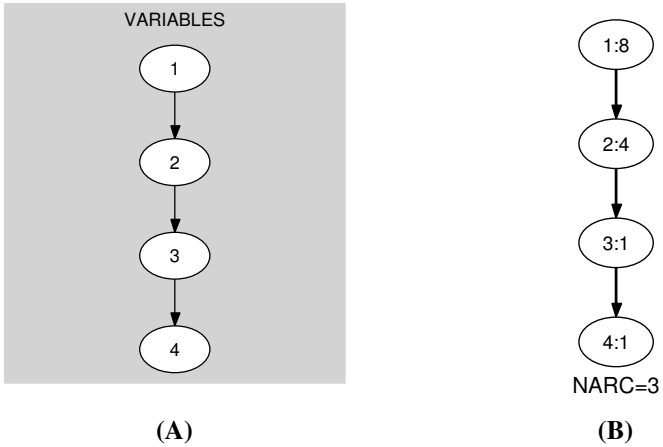


Figure 4.140: Initial and final graph of the `decreasing` constraint

Automaton Figure 4.141 depicts the automaton associated to the `decreasing` constraint. To each pair of consecutive variables $(\texttt{VAR}_i, \texttt{VAR}_{i+1})$ of the collection `VARIABLES` corresponds a 0-1 signature variable $\texttt{S}_i$. The following signature constraint links $\texttt{VAR}_i$, $\texttt{VAR}_{i+1}$ and $\texttt{S}_i$: $\texttt{VAR}_i < \texttt{VAR}_{i+1} \Leftrightarrow \texttt{S}_i$.

See also `strictly_decreasing`, `increasing`, `strictly_increasing`.

Key words decomposition, order constraint, automaton, automaton without counters, sliding cyclic(1) constraint network(1).

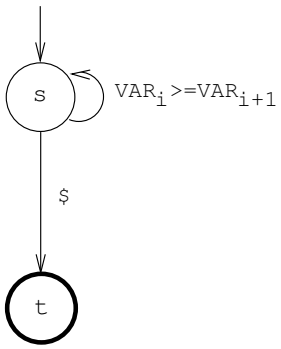


Figure 4.141: Automaton of the `decreasing` constraint

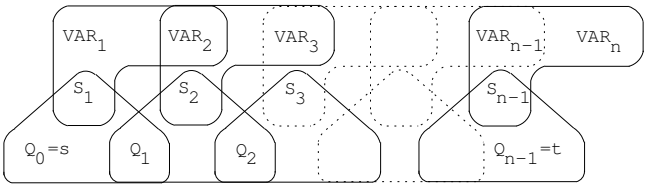


Figure 4.142: Hypergraph of the reformulation corresponding to the automaton of the `decreasing` constraint

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 $= V_1, \ldots, V_m$ is a *valley* if and only if there exist an i $(1 < i \leq k)$ such that $V_{i-1} > V_i$ and $V_i = V_{i+1} = \ldots = V_k$ 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

$$\texttt{deepest\_valley}\left(2, \left\{\begin{array}{l}\texttt{var}-5,\\ \texttt{var}-3,\\ \texttt{var}-4,\\ \texttt{var}-8,\\ \texttt{var}-8,\\ \texttt{var}-2,\\ \texttt{var}-7,\\ \texttt{var}-1\end{array}\right\}\right)$$

The previous constraint holds since 2 is the deepest valley of the sequence 5 3 4 8 8 2 7 1.

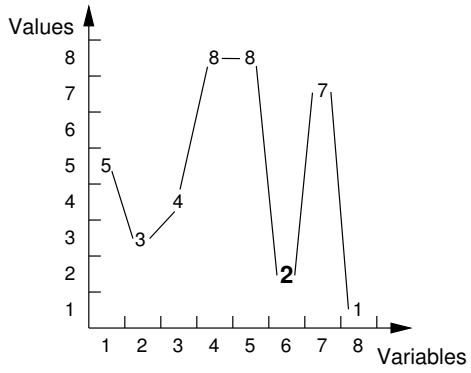


Figure 4.143: The sequence and its deepest valley

Automaton Figure 4.144 depicts the automaton associated to the deepest_valley constraint. To each pair of consecutive variables $(\texttt{VAR}_i, \texttt{VAR}_{i+1})$ of the collection VARIABLES corresponds a signature variable $\texttt{S}_i$. The following signature constraint links $\texttt{VAR}_i$, $\texttt{VAR}_{i+1}$ and $\texttt{S}_i$:

$$\texttt{VAR}_i < \texttt{VAR}_{i+1} \Leftrightarrow \texttt{S}_i = 0 \wedge \texttt{VAR}_i = \texttt{VAR}_{i+1} \Leftrightarrow \texttt{S}_i = 1 \wedge \texttt{VAR}_i > \texttt{VAR}_{i+1} \Leftrightarrow \texttt{S}_i = 2.$$

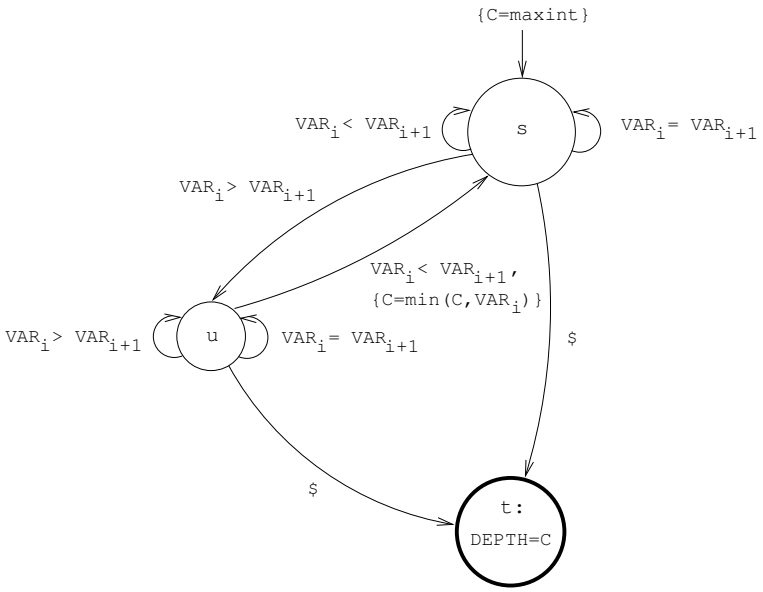


Figure 4.144: Automaton of the `deepest_valley` constraint

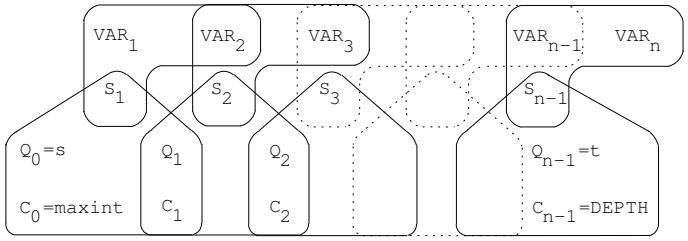


Figure 4.145: Hypergraph of the reformulation corresponding to the automaton of the `deepest_valley` constraint

See also `valley`, `heighest_peak`.

Key words sequence, maxint, automaton, automaton with counters, sliding cyclic(1) constraint network(2).

4.68 derangement

Origin Derived from cycle.

Constraint derangement(NODES)

Argument(s) NODES : collection(index − int, succ − dvar)

Restriction(s) required(NODES, [index, succ])
NODES.index ≥ 1
NODES.index ≤ |NODES|
distinct(NODES, index)
NODES.succ ≥ 1
NODES.succ ≤ |NODES|

Purpose Enforce to have a permutation with no cycle of length one. The permutation is depicted by the succ attribute of the NODES collection.

Arc input(s) NODES

Arc generator $CLIQUE \mapsto$ collection(nodes1, nodes2)

Arc arity 2

Arc constraint(s)
- nodes1.succ = nodes2.index
- nodes1.succ ≠ nodes1.index

Graph property(ies) $\mathbf{NTREE} = 0$

Example
$$\texttt{derangement}\left(\left\{\begin{array}{ll}\texttt{index}-1 & \texttt{succ}-2,\\ \texttt{index}-2 & \texttt{succ}-1,\\ \texttt{index}-3 & \texttt{succ}-5,\\ \texttt{index}-4 & \texttt{succ}-3,\\ \texttt{index}-5 & \texttt{succ}-4\end{array}\right\}\right)$$

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. $\mathbf{NTREE} = 0$).

Graph model In order to express the binary constraint that links two vertices of the NODES collection one has to make explicit the index value of the vertices. This is why the derangement 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.

Forbiding cycles of length one is achieved by the second condition of the arc constraint.

Signature Since 0 is the smallest possible value of $\mathbf{NTREE}$ we can rewrite the graph property $\mathbf{NTREE} = 0$ to $\mathbf{NTREE} \leq 0$. This leads to simplify $\overline{\mathbf{NTREE}}$ to $\underline{\mathbf{NTREE}}$.

Remark A special case of the `cycle` [37] constraint.

See also `alldifferent`, `cycle`.

Key words graph constraint, permutation.

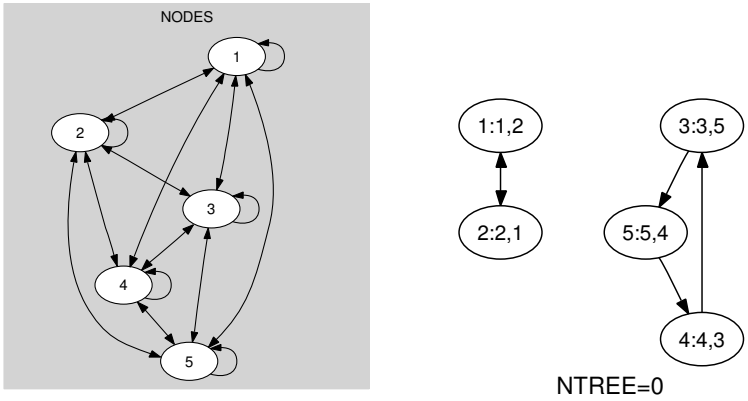


Figure 4.146: Initial and final graph of the `derangement` constraint

4.69 differ_from_at_least_k_pos

Origin Inspired by [56].

Constraint differ_from_at_least_k_pos(K, VECTOR1, VECTOR2)

Type(s) VECTOR : collection(var − dvar)

Argument(s)

K	:	int
VECTOR1	:	VECTOR
VECTOR2	:	VECTOR

Restriction(s) required(VECTOR, var)
K ≥ 0
K ≤ |VECTOR1|
|VECTOR1| = |VECTOR2|

Purpose Enforce two vectors VECTOR1 and VECTOR2 to differ from at least K positions.

Arc input(s) VECTOR1 VECTOR2

Arc generator $PRODUCT(=) \mapsto$ collection(vector1, vector2)

Arc arity 2

Arc constraint(s) vector1.var ≠ vector2.var

Graph property(ies) **NARC** ≥ K

Example differ_from_at_least_k_pos$\left(2, \left\{ \begin{array}{l} \mathtt{var}-2, \\ \mathtt{var}-5, \\ \mathtt{var}-2, \\ \mathtt{var}-0 \end{array} \right\}, \left\{ \begin{array}{l} \mathtt{var}-3, \\ \mathtt{var}-6, \\ \mathtt{var}-2, \\ \mathtt{var}-1 \end{array} \right\} \right)$

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 differ_from_at_least_k_pos constraint. Let $\mathtt{VAR1}_i$ and $\mathtt{VAR2}_i$ be the i^{th} variables of the VECTOR1 and VECTOR2 collections. To each pair of variables $(\mathtt{VAR1}_i, \mathtt{VAR2}_i)$ corresponds a signature variable $\mathtt{S}_i$. The following signature constraint links $\mathtt{VAR1}_i$, $\mathtt{VAR2}_i$ and $\mathtt{S}_i$: $\mathtt{VAR1}_i = \mathtt{VAR2}_i \Leftrightarrow \mathtt{S}_i$.

Remark Used in the **Arc constraint(s)** slot of the all_differ_from_at_least_k_pos constraint.

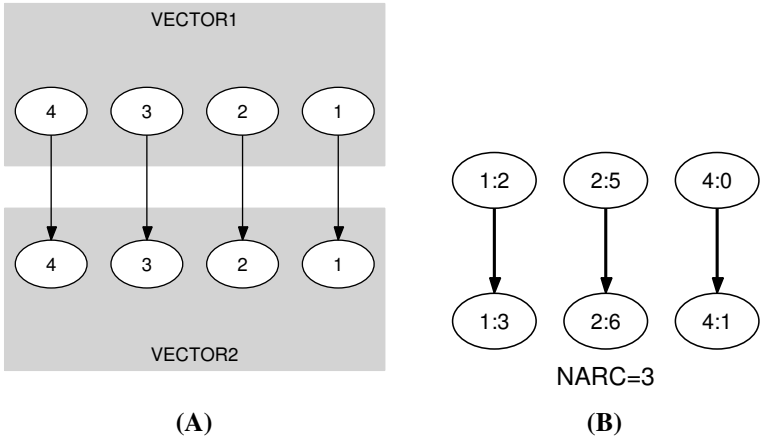


Figure 4.147: Initial and final graph of the `differ_from_at_least_k_pos` constraint

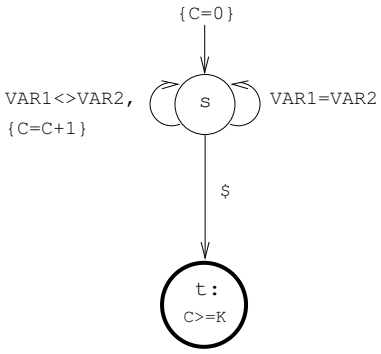


Figure 4.148: Automaton of the `differ_from_at_least_k_pos` constraint

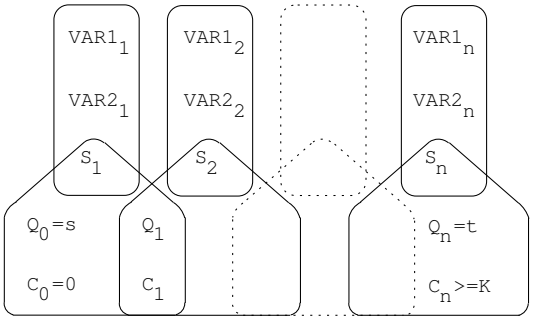


Figure 4.149: Hypergraph of the reformulation corresponding to the automaton of the `differ_from_at_least_k_pos` constraint

Used in `all_differ_from_at_least_k_pos`.

Key words value constraint, vector, automaton, automaton with counters, alpha-acyclic constraint network(2).

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) |ORTHOTOPE| > 0
require_at_least(2, ORTHOTOPE, [ori, siz, end])
ORTHOTOPE.siz ≥ 0
required(ORTHOTOPES, orth)
same_size(ORTHOTOPES, 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 $SELF \mapsto$ collection(orthotopes)

Arc arity 1

Arc constraint(s) orth_link_ori_siz_end(orthotopes.orth)

Graph property(ies) $\mathbf{NARC} = |\text{ORTHOTOPES}|$

Arc input(s) ORTHOTOPES

Arc generator $CLIQUE(\neq) \mapsto$ collection(orthotopes1, orthotopes2)

Arc arity 2

Arc constraint(s) two_orth_do_not_overlap(orthotopes1.orth, orthotopes2.orth)

Graph property(ies) $\mathbf{NARC} = |\text{ORTHOTOPES}| * |\text{ORTHOTOPES}| - |\text{ORTHOTOPES}|$

Example

$$\texttt{diffn}\left(\left\{\begin{array}{l}\texttt{orth} - \left\{\begin{array}{lll}\texttt{ori}-2 & \texttt{siz}-2 & \texttt{end}-4,\\ \texttt{ori}-1 & \texttt{siz}-3 & \texttt{end}-4\end{array}\right\},\\ \texttt{orth} - \left\{\begin{array}{lll}\texttt{ori}-4 & \texttt{siz}-4 & \texttt{end}-8,\\ \texttt{ori}-3 & \texttt{siz}-3 & \texttt{end}-3\end{array}\right\},\\ \texttt{orth} - \left\{\begin{array}{lll}\texttt{ori}-9 & \texttt{siz}-2 & \texttt{end}-11,\\ \texttt{ori}-4 & \texttt{siz}-3 & \texttt{end}-7\end{array}\right\}\end{array}\right\}\right)$$

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 $\mathbf{NARC}$ 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.

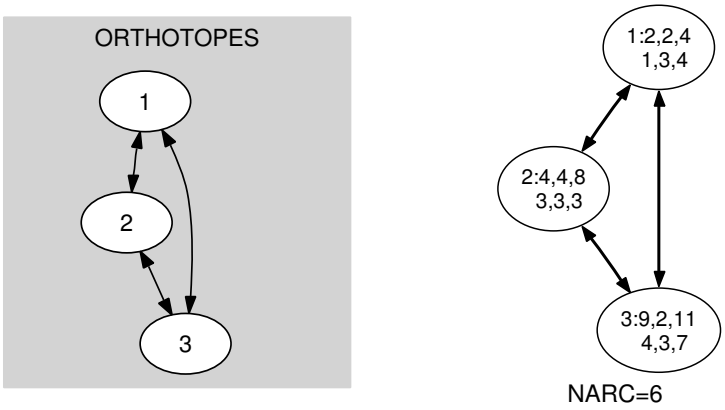


Figure 4.150: Initial and final graph of the `diffn` constraint

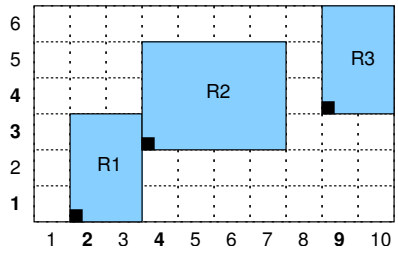


Figure 4.151: The three rectangles of the example

Graph model The `diffn` constraint is expressed by using two graph constraints:

- The first graph constraint enforces for each dimension and for each orthotope the link between the corresponding `ori`, `siz` and `end` attributes.
- The second graph constraint imposes each pair of distinct orthotopes to not overlap.

Signature Since |`ORTHOTOPES`| is the maximum number of vertices of the final graph of the first graph constraint we can rewrite $\mathbf{NARC} = |\texttt{ORTHOTOPES}|$ to $\mathbf{NARC} \geq |\texttt{ORTHOTOPES}|$. This leads to simplify $\underline{\overline{\mathbf{NARC}}}$ to $\overline{\mathbf{NARC}}$.

Since we use the $CLIQUE(\neq)$ arc generator on the `ORTHOTOPES` collection, $|\texttt{ORTHOTOPES}| \cdot |\texttt{ORTHOTOPES}| - |\texttt{ORTHOTOPES}|$ is the maximum number of vertices of the final graph of the second graph constraint. Therefore we can rewrite $\mathbf{NARC} = |\texttt{ORTHOTOPES}| \cdot |\texttt{ORTHOTOPES}| - |\texttt{ORTHOTOPES}|$ to $\mathbf{NARC} \geq |\texttt{ORTHOTOPES}| \cdot |\texttt{ORTHOTOPES}| - |\texttt{ORTHOTOPES}|$. Again, this leads to simplify $\underline{\overline{\mathbf{NARC}}}$ to $\overline{\mathbf{NARC}}$.

Usage The `diffn` constraint occurs in placement and scheduling problems. It was for instance used for scheduling problems where one has to both assign each non-premptive 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 `diffn` a possible filtering algorithm based on *sweep* is described in [97]. For the n-dimensional case of `diffn` 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 *squared squares* problem [100] (based on the fact that no waste is permitted) are given in [101] and in [102].

Used in `diffn_column`, `diffn_include`, `place_in_pyramid`.

See also `orth_link_ori_siz_end`, `two_orth_do_not_overlap`.

Key words decomposition, geometrical constraint, orthotope, polygon, non-overlapping, sweep, squared squares.

4.71 diffn_column

Origin CHIP: option guillotine cut (column) of diffn.

Constraint diffn_column(ORTHOTOPES, N)

Type(s) ORTHOTOPE : collection(ori − dvar, siz − dvar, end − dvar)

Argument(s)

ORTHOTOPES	:	collection(orth − ORTHOTOPE)
N	:	int

Restriction(s)
- |ORTHOTOPE| > 0
- require_at_least(2, ORTHOTOPE, [ori, siz, end])
- ORTHOTOPE.siz ≥ 0
- required(ORTHOTOPES, orth)
- same_size(ORTHOTOPES, orth)
- N > 0
- N ≤ |ORTHOTOPE|
- diffn(ORTHOTOPES)

Purpose □

Arc input(s) ORTHOTOPES

Arc generator $CLIQUE(<) \mapsto$ collection(orthotopes1, orthotopes2)

Arc arity 2

Arc constraint(s) two_orth_column(orthotopes1.orth, orthotopes2.orth, N)

Graph property(ies) $\mathbf{NARC} = |\text{ORTHOTOPES}| * (|\text{ORTHOTOPES}| - 1)/2$

Example

$$\text{diffn\_column}\left(\left\{\begin{array}{lll}\text{orth} - \{\text{ori} - 1 & \text{siz} - 3 & \text{end} - 4,\\ \text{ori} - 1 & \text{siz} - 1 & \text{end} - 2\}\\ \text{orth} - \{\text{ori} - 4 & \text{siz} - 2 & \text{end} - 6,\\ \text{ori} - 1 & \text{siz} - 3 & \text{end} - 4\}\end{array}\right\}, 1\right)$$

See also diffn, two_orth_column, diffn_include.

Key words decomposition, geometrical constraint, positioning constraint, orthotope, guillotine cut.

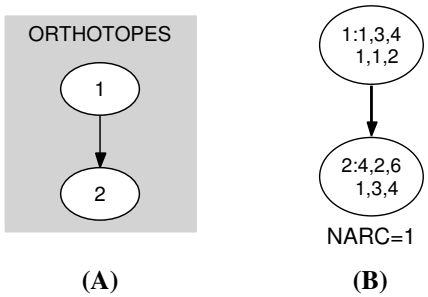


Figure 4.152: Initial and final graph of the `diffn_column` constraint

4.72 diffn_include

Origin CHIP: option guillotine cut (include) of diffn.

Constraint diffn_include(ORTHOTOPES, N)

Type(s)
ORTHOTOPE : collection(ori − dvar, siz − dvar, end − dvar)

Argument(s)
ORTHOTOPES : collection(orth − ORTHOTOPE)
N : int

Restriction(s)
|ORTHOTOPE| > 0
require_at_least(2, ORTHOTOPE, [ori, siz, end])
ORTHOTOPE.siz ≥ 0
required(ORTHOTOPES, orth)
same_size(ORTHOTOPES, orth)
N > 0
N ≤ |ORTHOTOPE|
diffn(ORTHOTOPES)

Purpose □

Arc input(s) ORTHOTOPES

Arc generator $CLIQUE(<) \mapsto$ collection(orthotopes1, orthotopes2)

Arc arity 2

Arc constraint(s) two_orth_include(orthotopes1.orth, orthotopes2.orth, N)

Graph property(ies) $\mathbf{NARC} = |\text{ORTHOTOPES}| * (|\text{ORTHOTOPES}| - 1)/2$

Example
$$\text{diffn\_include}\left(\left\{\begin{array}{l}\text{orth}-\left\{\begin{array}{lll}\text{ori}-1 & \text{siz}-3 & \text{end}-4,\\ \text{ori}-1 & \text{siz}-1 & \text{end}-2\end{array}\right\},\\ \text{orth}-\left\{\begin{array}{lll}\text{ori}-1 & \text{siz}-2 & \text{end}-3,\\ \text{ori}-2 & \text{siz}-3 & \text{end}-5\end{array}\right\}\end{array}\right\}, 1\right)$$

See also diffn, two_orth_include, diffn_column.

Key words decomposition, geometrical constraint, positioning constraint, orthotope.

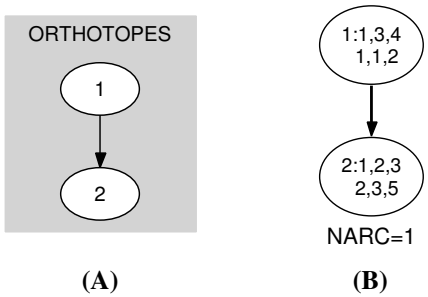


Figure 4.153: Initial and final graph of the `diffn_include` constraint

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)
$\mathtt{K} \geq 0$
$\mathtt{K} \leq |\mathtt{VARIABLES}|$

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 $SELF \mapsto$ collection(variables)

Arc arity 1

Arc constraint(s) in_set(variables.var, variables.bad)

Graph property(ies) $\mathbf{NARC} = \mathtt{K}$

Example

$$\mathtt{discrepancy}\left(\left\{\begin{array}{ll}\mathtt{var}-4 & \mathtt{bad}-\{1,4,6\},\\ \mathtt{var}-5 & \mathtt{bad}-\{0,1\},\\ \mathtt{var}-5 & \mathtt{bad}-\{1,6,9\},\\ \mathtt{var}-4 & \mathtt{bad}-\{1,4\},\\ \mathtt{var}-1 & \mathtt{bad}-\emptyset\end{array}\right\},2\right)$$

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.

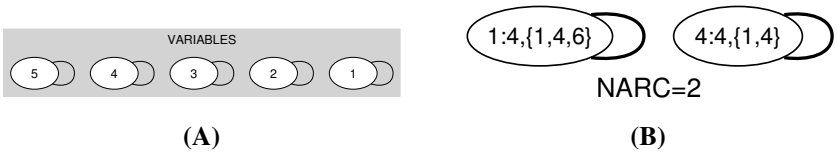


Figure 4.154: Initial and final graph of the discrepancy constraint

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 explictely 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 disjoint(VARIABLES1, VARIABLES2)

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.

Arc input(s) VARIABLES1 VARIABLES2

Arc generator $PRODUCT \mapsto$ collection(variables1, variables2)

Arc arity 2

Arc constraint(s) variables1.var = variables2.var

Graph property(ies) $\mathbf{NARC} = 0$

Example

$$\texttt{disjoint}\left(\begin{array}{l}\{\texttt{var}-1,\texttt{var}-9,\texttt{var}-1,\texttt{var}-5\},\\ \left\{\begin{array}{l}\texttt{var}-2,\\ \texttt{var}-7,\\ \texttt{var}-7,\\ \texttt{var}-0,\\ \texttt{var}-6,\\ \texttt{var}-8\end{array}\right\}\end{array}\right)$$

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 disjoint constraint holds. Figure 4.155 shows the initial graph. Since we use the $\mathbf{NARC} = 0$ graph property the final graph is empty.

Graph model $PRODUCT$ 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 $\mathbf{NARC} = 0$ the final graph will be empty.

Signature Since 0 is the smallest number of arcs of the final graph we can rewrite $\mathbf{NARC} = 0$ to $\mathbf{NARC} \leq 0$. This leads to simplify $\overline{\mathbf{NARC}}$ to $\underline{\mathbf{NARC}}$.

Automaton Figure 4.156 depicts the automaton associated to the disjoint constraint. To each variable $\texttt{VAR1}_i$ of the collection VARIABLES1 corresponds a signature variable $\texttt{S}_i$, which is equal to 0. To each variable $\texttt{VAR2}_i$ of the collection VARIABLES2 corresponds a signature variable $\texttt{S}_{i+|\texttt{VARIABLES1}|}$, which is equal to 1.

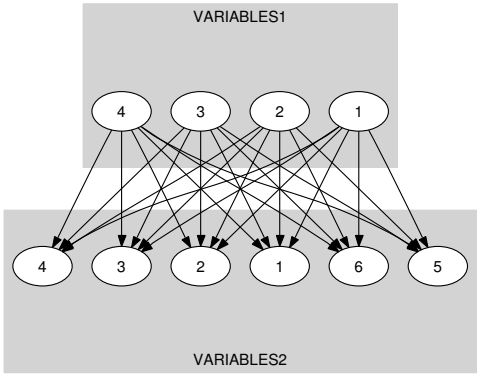


Figure 4.155: Initial graph of the `disjoint` constraint (the final graph is empty)

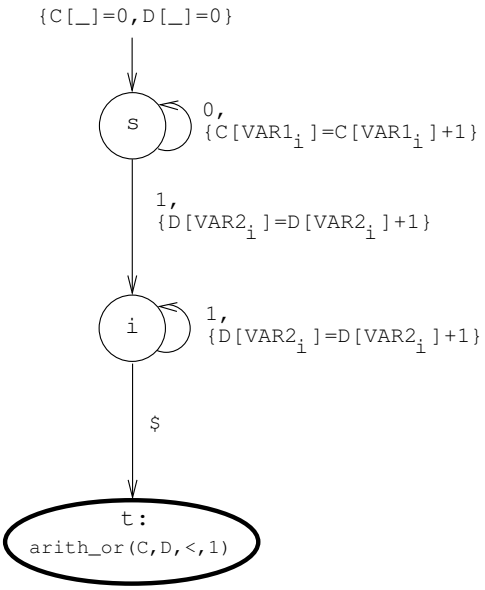


Figure 4.156: Automaton of the `disjoint` constraint

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 bee 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.

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`.

Arc input(s) `TASKS1`

Arc generator $SELF \mapsto$ `collection`(`tasks1`)

Arc arity 1

Arc constraint(s) `tasks1.origin` + `tasks1.duration` = `tasks1.end`

Graph property(ies) $\mathbf{NARC} = |$`TASKS1`$|$

Arc input(s) `TASKS2`

Arc generator $SELF \mapsto$ `collection`(`tasks2`)

Arc arity 1

Arc constraint(s) `tasks2.origin` + `tasks2.duration` = `tasks2.end`

Graph property(ies) $\mathbf{NARC} = |$`TASKS2`$|$

Arc input(s) `TASKS1` `TASKS2`

Arc generator $PRODUCT \mapsto$ `collection`(`tasks1`, `tasks2`)

Arc arity 2

Arc constraint(s)

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

Graph property(ies) $\mathbf{NARC} = 0$

Example

$$\texttt{disjoint\_tasks}\left(\begin{array}{l}\left\{\begin{array}{lll}\texttt{origin}-6 & \texttt{duration}-5 & \texttt{end}-11,\\ \texttt{origin}-8 & \texttt{duration}-2 & \texttt{end}-10\end{array}\right\},\\ \left\{\begin{array}{lll}\texttt{origin}-2 & \texttt{duration}-2 & \texttt{end}-4,\\ \texttt{origin}-3 & \texttt{duration}-3 & \texttt{end}-6,\\ \texttt{origin}-12 & \texttt{duration}-1 & \texttt{end}-13\end{array}\right\}\end{array}\right)$$

Figure 4.157 shows the initial graph of the third graph constraint. Because of the graph property $\mathbf{NARC} = 0$ 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.

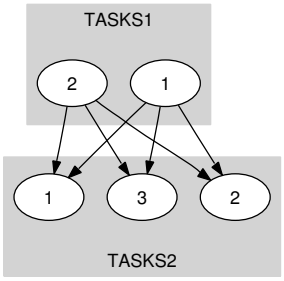


Figure 4.157: Initial graph of the disjoint_tasks constraint (the final graph is empty)

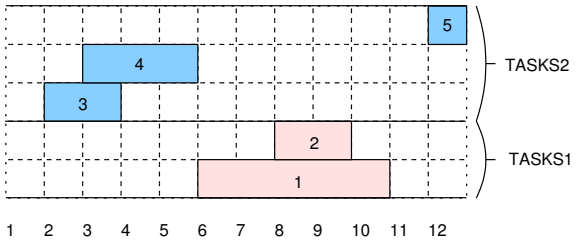


Figure 4.158: Fixed tasks of the disjoint_tasks constraint

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 $\mathbf{NARC} = 0$ 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 $\mathbf{NARC} = |\texttt{TASKS1}|$. This leads to simplify $\overline{\mathbf{NARC}}$ to $\overline{\mathbf{NARC}}$.

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 $\mathbf{NARC} = 0$ to $\mathbf{NARC} \leq 0$. This leads to simplify $\underline{\overline{\mathbf{NARC}}}$ to $\underline{\mathbf{NARC}}$.

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.

4.76 disjunctive

Origin [107]

Constraint disjunctive(TASKS)

Synonym(s) one_machine.

Argument(s) TASKS : collection(origin − dvar, duration − dvar)

Restriction(s) required(TASKS, [origin, duration])
TASKS.duration ≥ 0

Purpose All the tasks of the collection TASKS should not overlap.

Arc input(s) TASKS

Arc generator $CLIQUE(<) \mapsto$ collection(tasks1, tasks2)

Arc arity 2

Arc constraint(s)
$$\bigvee \left(\begin{array}{l} \texttt{tasks1.duration} = 0, \\ \texttt{tasks2.duration} = 0, \\ \texttt{tasks1.origin} + \texttt{tasks1.duration} \leq \texttt{tasks2.origin}, \\ \texttt{tasks2.origin} + \texttt{tasks2.duration} \leq \texttt{tasks1.origin} \end{array} \right)$$

Graph property(ies) $\mathbf{NARC} = |\texttt{TASKS}| * (|\texttt{TASKS}| - 1)/2$

Example
$$\texttt{disjunctive} \left(\left\{ \begin{array}{ll} \texttt{origin} - 1 & \texttt{duration} - 3, \\ \texttt{origin} - 2 & \texttt{duration} - 0, \\ \texttt{origin} - 7 & \texttt{duration} - 2, \\ \texttt{origin} - 4 & \texttt{duration} - 1 \end{array} \right\} \right)$$

Parts (A) and (B) of Figure 4.159 respectively show the initial and final graph. The disjunctive constraint holds since all the arcs of the initial graph belong to the final graph: all the non-overlapping constraints holds.

Graph model We generate a *clique* 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 disjunctive constraint are described in [109] and [110].

See also cumulative, diffn.

Key words scheduling constraint, resource constraint, decomposition.

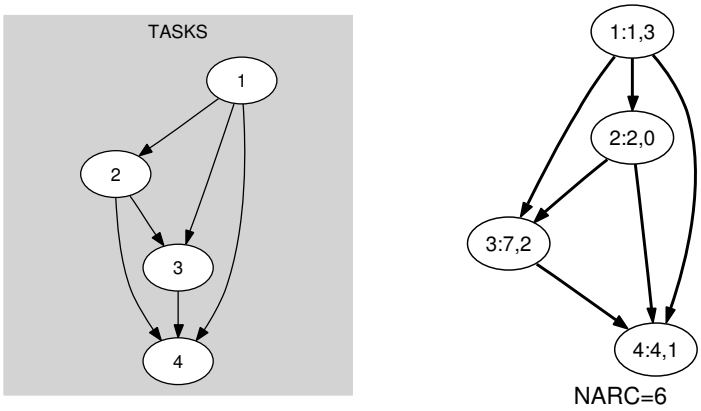


Figure 4.159: Initial and final graph of the `disjunctive` constraint

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 $\leq$ |VARIABLES1| $*$ |VARIABLES2|
required(VARIABLES1, var)
required(VARIABLES2, var)
|VARIABLES1| = |VARIABLES2|
CTR $\in [=, \neq, <, \geq, >, \leq]$

Purpose

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:

- 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.

Arc input(s) VARIABLES1/ VARIABLES2

Arc generator $CLIQUE(\neq) \mapsto$ collection(variables1, variables2)

Arc arity 2

Arc constraint(s) variables1.var CTR variables2.var

Graph property(ies) **DISTANCE** = DIST

Example

$$\texttt{distance\_between}\left(\begin{array}{l} 2, \left\{\begin{array}{l}\texttt{var}-3,\\ \texttt{var}-4,\\ \texttt{var}-6,\\ \texttt{var}-2,\\ \texttt{var}-4\end{array}\right\},\\ \left\{\begin{array}{l}\texttt{var}-2,\\ \texttt{var}-6,\\ \texttt{var}-9,\\ \texttt{var}-3,\\ \texttt{var}-6\end{array}\right\}, < \end{array}\right)$$

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.

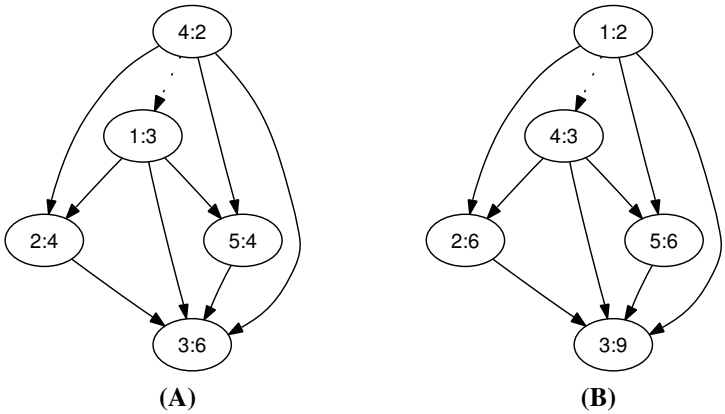


Figure 4.160: Final graphs of the distance_between constraint

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 ≥ 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 ($1 \leq i < n$):

- 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 $PATH \mapsto$ collection(variables1, variables2)

Arc arity 2

Arc constraint(s) variables1.var CTR variables2.var

Graph property(ies) **DISTANCE** = DIST

Example

$$\texttt{distance\_change}\left(\begin{array}{l} 1, \left\{\begin{array}{l}\texttt{var}-3,\\ \texttt{var}-3,\\ \texttt{var}-1,\\ \texttt{var}-2,\\ \texttt{var}-2\end{array}\right\},\\ \left\{\begin{array}{l}\texttt{var}-4,\\ \texttt{var}-4,\\ \texttt{var}-3,\\ \texttt{var}-3,\\ \texttt{var}-3\end{array}\right\}, \neq \end{array}\right)$$

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 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.

Automaton Figure 4.162 depicts the automaton associated to the distance_change constraint. Let $(\texttt{VAR1}_i, \texttt{VAR1}_{i+1})$ and $(\texttt{VAR2}_i, \texttt{VAR2}_{i+1})$ respectively be the i^{th} pairs of consecutive variables of the collections VARIABLES1 and VARIABLES2. To each quadruple $(\texttt{VAR1}_i, \texttt{VAR1}_{i+1}, \texttt{VAR2}_i, \texttt{VAR2}_{i+1})$ corresponds a 0-1 signature variable $\texttt{S}_i$. The following signature constraint links these variables:

$((\texttt{VAR1}_i = \texttt{VAR1}_{i+1}) \wedge (\texttt{VAR2}_i \neq \texttt{VAR2}_{i+1})) \vee$

$((\texttt{VAR1}_i \neq \texttt{VAR1}_{i+1}) \wedge (\texttt{VAR2}_i = \texttt{VAR2}_{i+1})) \Leftrightarrow \texttt{S}_i.$

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

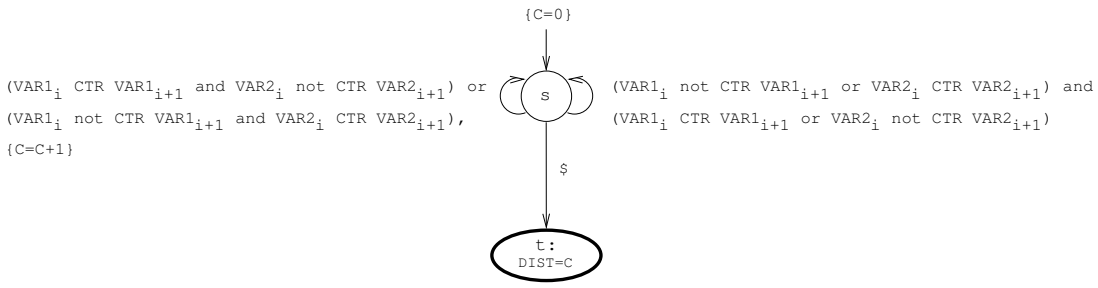


Figure 4.162: Automaton of the `distance_change` constraint

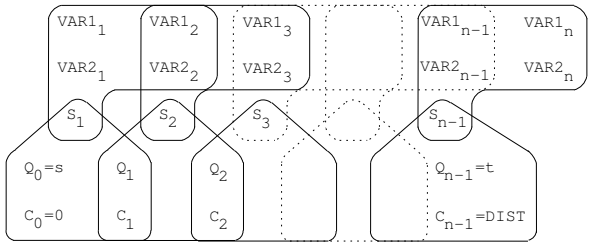


Figure 4.163: Hypergraph of the reformulation corresponding to the automaton of the `distance_change` constraint

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) col(VALUE − collection(var01 − int, value − dvar), [item(var01 − 1, value − VAR)])

Arc input(s) VALUE VALUES

Arc generator $PRODUCT \mapsto$ collection(value, values)

Arc arity 2

Arc constraint(s) value.value = values.value ⇔ values.var01 = 1

Graph property(ies) $\mathbf{NARC} = |$VALUES$|$

Example domain_constraint$\left(5, \left\{ \begin{array}{ll} \texttt{var01} - 0 & \texttt{value} - 9, \\ \texttt{var01} - 1 & \texttt{value} - 5, \\ \texttt{var01} - 0 & \texttt{value} - 2, \\ \texttt{var01} - 0 & \texttt{value} - 7 \end{array} \right\} \right)$

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 $\{\texttt{var01} - 1\ \texttt{value} - \texttt{VAR}\}$ 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 VALUES the maximum number of arcs of the final graph is also equal to VALUES. Therefore we can rewrite the graph property $\mathbf{NARC} = |\texttt{VALUES}|$ to $\mathbf{NARC} \geq |\texttt{VALUES}|$. This leads to simplify $\underline{\overline{\mathbf{NARC}}}$ to $\overline{\mathbf{NARC}}$.

Automaton Figure 4.165 depicts the automaton associated to the domain_constraint constraint. Let $\texttt{VAR01}_i$ and $\texttt{VALUE}_i$ respectively be the var01 and the value attributes of the i^{th} item of the VALUES collection. To each triple $(\texttt{VAR}, \texttt{VAR01}_i, \texttt{VALUE}_i)$ corresponds a 0-1 signature variable $\texttt{S}_i$ as well as the following signature constraint: $((\texttt{VAR} = \texttt{VALUE}_i) \Leftrightarrow \texttt{VAR01}_i) \Leftrightarrow \texttt{S}_i$.

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 link_set_to_booleans.

Key words decomposition, channeling constraint, domain channel, boolean channel, linear programming, automaton, automaton without counters, centered cyclic(1) constraint network(1), derived collection.

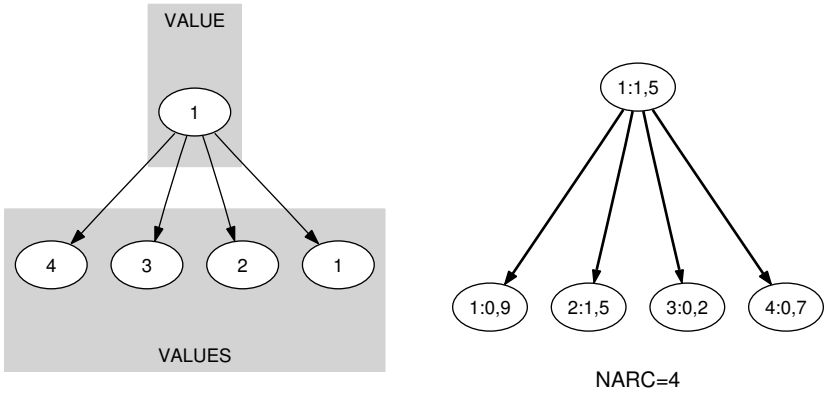


Figure 4.164: Initial and final graph of the domain_constraint constraint

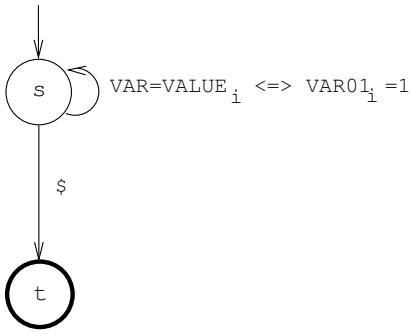


Figure 4.165: Automaton of the domain_constraint constraint

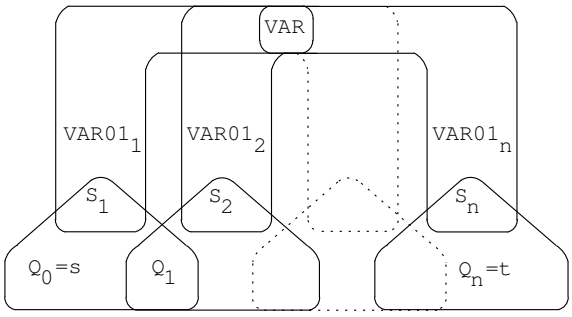


Figure 4.166: Hypergraph of the reformulation corresponding to the automaton of the domain_constraint constraint

4.80 elem

Origin Derived from element.

Constraint elem(ITEM, TABLE)

Usual name element

Argument(s)

ITEM	:	collection(index − dvar, value − dvar)
TABLE	:	collection(index − int, value − dvar)

Restriction(s)

required(ITEM, [index, value])
ITEM.index ≥ 1
ITEM.index $\leq$ |TABLE|
|ITEM| $= 1$
required(TABLE, [index, value])
TABLE.index ≥ 1
TABLE.index $\leq$ |TABLE|
distinct(TABLE, index)

Purpose ITEM is equal to one of the entries of the table TABLE.

Arc input(s) ITEM TABLE

Arc generator $PRODUCT \mapsto$ collection(item, table)

Arc arity 2

Arc constraint(s)

- item.index = table.index
- item.value = table.value

Graph property(ies) $\mathbf{NARC} = 1$

Example

$$\texttt{elem}\left(\begin{array}{l}\{\texttt{index}-3\ \texttt{value}-2\},\\ \left\{\begin{array}{ll}\texttt{index}-1 & \texttt{value}-6,\\ \texttt{index}-2 & \texttt{value}-9,\\ \texttt{index}-3 & \texttt{value}-2,\\ \texttt{index}-4 & \texttt{value}-9\end{array}\right\}\end{array}\right)$$

Parts (A) and (B) of Figure 4.167 respectively show the initial and final graph. Since we use the $\mathbf{NARC}$ graph property, the unique arc of the final graph is stressed in bold.

Graph model We regroup the INDEX and VALUE parameters of the original element constraint element(INDEX, TABLE, VALUE) into the parameter ITEM. We also make explicit the different indices of the table TABLE.

Signature Since all the index attributes of TABLE 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 $\mathbf{NARC} = 1$ to $\mathbf{NARC} \geq 1$ and simplify $\overline{\mathbf{NARC}}$ to $\overline{\mathbf{NARC}}$.

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 $\texttt{INDEX}_i$ and $\texttt{VALUE}_i$ respectively be the `index` and the `value` attributes of the i^{th} item of the `TABLE` collection. To each quadruple $(\texttt{INDEX}, \texttt{VALUE}, \texttt{INDEX}_i, \texttt{VALUE}_i)$ corresponds a 0-1 signature variable $\texttt{S}_i$ as well as the following signature constraint: $((\texttt{INDEX} = \texttt{INDEX}_i) \wedge (\texttt{VALUE} = \texttt{VALUE}_i)) \Leftrightarrow \texttt{S}_i$.

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:

$$\texttt{elem}\left(\begin{array}{l}\left\{\begin{array}{ll}\texttt{index} - \texttt{Machine} & \texttt{value} - \texttt{Duration}\end{array}\right\}, \\ \left\{\begin{array}{ll}\texttt{index} - 1 & \texttt{value} - \texttt{Dur}_1, \\ \texttt{index} - 2 & \texttt{value} - \texttt{Dur}_2, \\ & \vdots \\ \texttt{index} - m & \texttt{value} - \texttt{Dur}_m\end{array}\right\}\end{array}\right)$$

   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,
   - $\texttt{Dur}_1, \texttt{Dur}_2, \ldots, \texttt{Dur}_m$ 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:

$$\texttt{elem}\left(\begin{array}{l}\left\{\begin{array}{ll}\texttt{index} - \texttt{Next}_i & \texttt{value} - \texttt{distance}_i\end{array}\right\}, \\ \left\{\begin{array}{ll}\texttt{index} - 1 & \texttt{value} - \texttt{Dist}_{i_1}, \\ \texttt{index} - 2 & \texttt{value} - \texttt{Dist}_{i_2}, \\ & \vdots \\ \texttt{index} - m & \texttt{value} - \texttt{Dist}_{i_m}\end{array}\right\}\end{array}\right)$$

   where:

   - $\texttt{Next}_i$ is a domain variable which gives the index of the location the vehicle will visit just after the i^{th} location,
   - $\texttt{distance}_i$ is a domain variable which corresponds to the distance between location i and the location the vehicle will visit just after,
   - $\texttt{Dist}_{i_1}, \texttt{Dist}_{i_2}, \ldots, \texttt{Dist}_{i_m}$ are the respective distances between location i and locations $1, 2, \ldots, m$.

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:

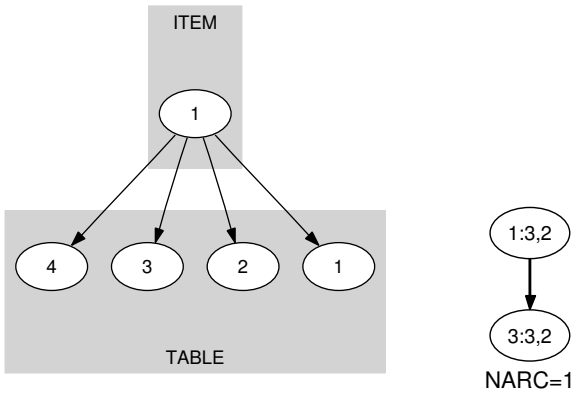


Figure 4.167: Initial and final graph of the elem constraint

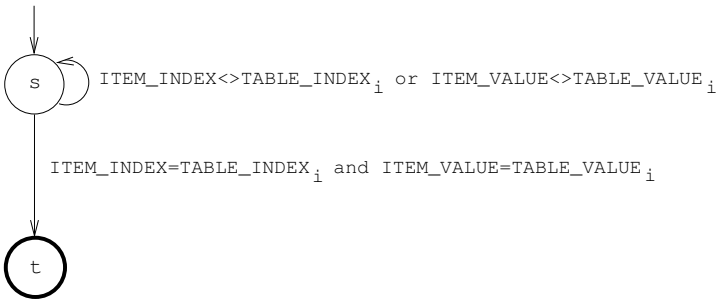


Figure 4.168: Automaton of the elem constraint

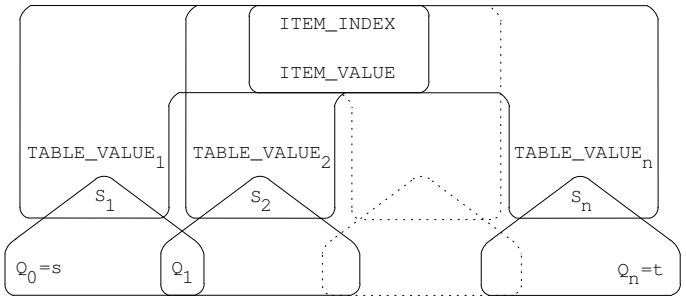


Figure 4.169: Hypergraph of the reformulation corresponding to the automaton of the elem constraint

$$\texttt{elem}\left(\begin{array}{l}\left\{\begin{array}{ll}\texttt{index}-\texttt{Choice} & \texttt{value}-\texttt{Cost}\end{array}\right\},\\ \left\{\begin{array}{ll}\texttt{index}-1 & \texttt{value}-\texttt{Cost}_1,\\ \texttt{index}-2 & \texttt{value}-\texttt{Cost}_2,\\ & \vdots\\ \texttt{index}-m & \texttt{value}-\texttt{Cost}_m\end{array}\right\}\end{array}\right)$$

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,
- $\texttt{Cost}_1, \texttt{Cost}_2, \ldots, \texttt{Cost}_m$ are the respective costs associated to the alternatives $1, 2, \ldots, m$.

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 ≥ 1
INDEX $\leq |$TABLE$|$
required(TABLE, value)

Purpose VALUE is equal to the INDEXth item of TABLE.

Derived Collection(s) $\texttt{col}\left(\begin{array}{l}\texttt{ITEM} - \texttt{collection}(\texttt{index} - \texttt{dvar}, \texttt{value} - \texttt{dvar}),\\ [\texttt{item}(\texttt{index} - \texttt{INDEX}, \texttt{value} - \texttt{VALUE})]\end{array}\right)$

Arc input(s) ITEM TABLE

Arc generator $PRODUCT \mapsto \texttt{collection}(\texttt{item}, \texttt{table})$

Arc arity 2

Arc constraint(s)

- item.index = table.key
- item.value = table.value

Graph property(ies) $\mathbf{NARC} = 1$

Example $\texttt{element}\left(3, \left\{\begin{array}{l}\texttt{value} - 6,\\ \texttt{value} - 9,\\ \texttt{value} - 2,\\ \texttt{value} - 9\end{array}\right\}, 2\right)$

Parts (A) and (B) of Figure 4.170 respectively show the initial and final graph. Since we use the $\mathbf{NARC}$ graph property, the unique arc of the final graph is stressed in bold.

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 $\mathbf{NARC} = 1$ to $\mathbf{NARC} \geq 1$ and simplify $\overline{\mathbf{NARC}}$ to $\overline{\mathbf{NARC}}$.

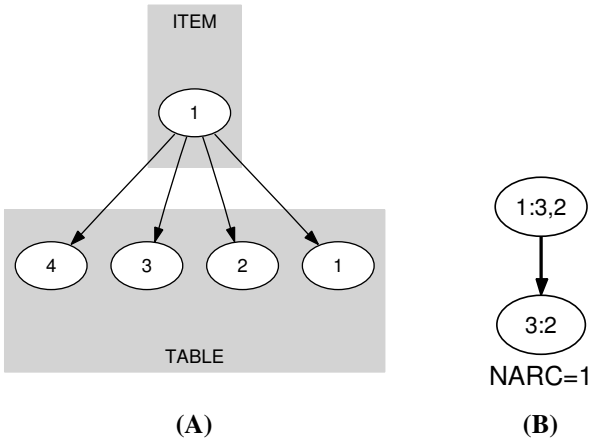


Figure 4.170: Initial and final graph of the `element` constraint

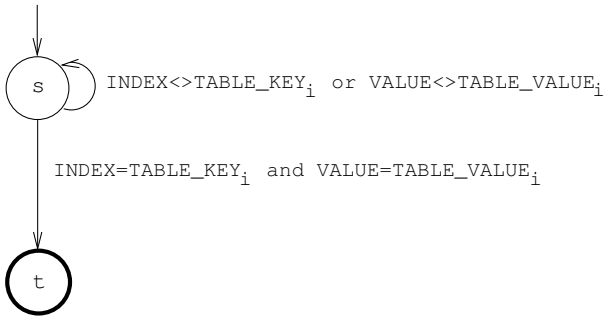


Figure 4.171: Automaton of the `element` constraint

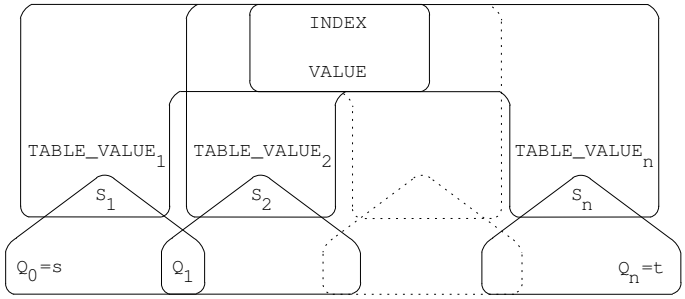


Figure 4.172: Hypergraph of the reformulation corresponding to the automaton of the `element` constraint

Automaton Figure 4.171 depicts the automaton associated to the element constraint. Let $\texttt{VALUE}_i$ be the value attribute of the i^{th} item of the TABLE collection. To each triple $(\texttt{INDEX}, \texttt{VALUE}, \texttt{VALUE}_i)$ corresponds a 0-1 signature variable $\texttt{S}_i$ as well as the following signature constraint: $(\texttt{INDEX} = i \wedge \texttt{VALUE} = \texttt{VALUE}_i) \Leftrightarrow \texttt{S}_i$.

Usage See elem.

Remark In the original element constraint of CHIP the index 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 element constraint, where the table is replaced by a directed acyclic graph describing the set of solutions.

See also elem, 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), derived collection.

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 $\leq |$TABLE$|$
$|$ITEM$| = 1$
required(TABLE, [index, value])
TABLE.index ≥ 1
TABLE.index $\leq |$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.

Arc input(s) ITEM TABLE

Arc generator $PRODUCT \mapsto$ collection(item, table)

Arc arity 2

Arc constraint(s)

- item.index = table.index
- item.value $\geq$ table.value

Graph property(ies) $\mathbf{NARC} = 1$

Example

$$\text{element\_greatereq}\left(\begin{array}{l}\{\text{index}-1\ \text{value}-8\},\\ \left\{\begin{array}{ll}\text{index}-1 & \text{value}-6,\\ \text{index}-2 & \text{value}-9,\\ \text{index}-3 & \text{value}-2,\\ \text{index}-4 & \text{value}-9\end{array}\right\}\end{array}\right)$$

Parts (A) and (B) of Figure 4.173 respectively show the initial and final graph. Since we use the $\mathbf{NARC}$ graph property, the unique arc of the final graph is stressed in bold.

Graph model Similar to the element constraint except that the *equality* constraint of the second condition of the arc constraint is replaced by a *greater than or equal to* 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 $\mathbf{NARC} = 1$ to $\mathbf{NARC} \geq 1$ and simplify $\overline{\mathbf{NARC}}$ to $\overline{\mathbf{NARC}}$.

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 $\texttt{INDEX}_i$ and $\texttt{VALUE}_i$ respectively be the `index` and the `value` attributes of the i^{th} item of the `TABLE` collection. To each quadruple $(\texttt{INDEX}, \texttt{VALUE}, \texttt{INDEX}_i, \texttt{VALUE}_i)$ corresponds a 0-1 signature variable $\texttt{S}_i$ as well as the following signature constraint: $((\texttt{INDEX} = \texttt{INDEX}_i) \wedge (\texttt{VALUE} \geq \texttt{VALUE}_i)) \Leftrightarrow \texttt{S}_i$.

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).

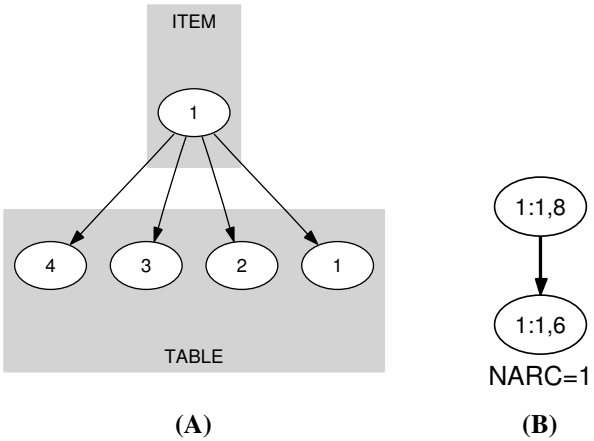


Figure 4.173: Initial and final graph of the element_greatereq constraint

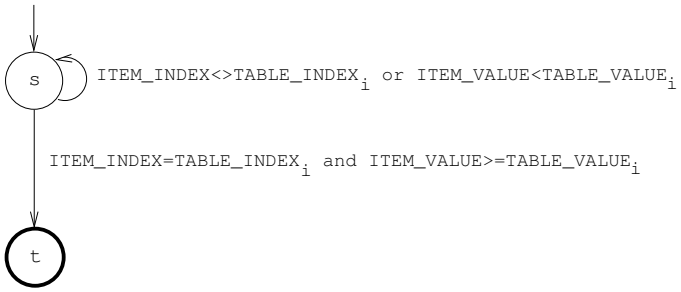


Figure 4.174: Automaton of the element_greatereq constraint

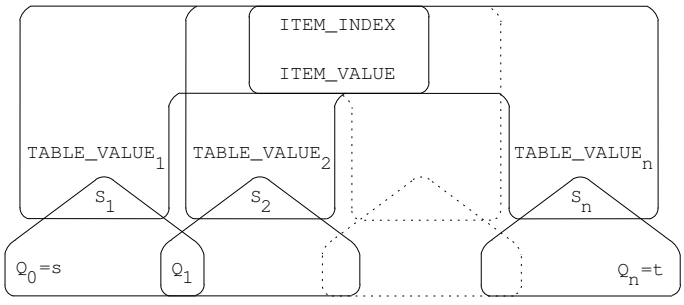


Figure 4.175: Hypergraph of the reformulation corresponding to the automaton of the element_greatereq constraint

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 $\leq$ |TABLE|
|ITEM| $= 1$
required(TABLE, [index, value])
TABLE.index ≥ 1
TABLE.index $\leq$ |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.

Arc input(s) ITEM TABLE

Arc generator $PRODUCT \mapsto$ collection(item, table)

Arc arity 2

Arc constraint(s)

- item.index = table.index
- item.value $\leq$ table.value

Graph property(ies) $\mathbf{NARC} = 1$

Example

$$\texttt{element\_lesseq}\left(\begin{array}{l}\{\texttt{index}-3\ \texttt{value}-1\},\\ \left\{\begin{array}{ll}\texttt{index}-1 & \texttt{value}-6,\\ \texttt{index}-2 & \texttt{value}-9,\\ \texttt{index}-3 & \texttt{value}-2,\\ \texttt{index}-4 & \texttt{value}-9\end{array}\right\}\end{array}\right)$$

Parts (A) and (B) of Figure 4.176 respectively show the initial and final graph. Since we use the $\mathbf{NARC}$ graph property, the unique arc of the final graph is stressed in bold.

Graph model Similar to the element constraint except that the *equality* constraint of the second condition of the arc constraint is replaced by a *less than or equal to* 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 $\mathbf{NARC} = 1$ to $\mathbf{NARC} \geq 1$ and simplify $\overline{\mathbf{NARC}}$ to $\overline{\mathbf{NARC}}$.

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 $\texttt{INDEX}_i$ and $\texttt{VALUE}_i$ respectively be the `index` and the `value` attributes of the i^{th} item of the `TABLE` collection. To each quadruple $(\texttt{INDEX}, \texttt{VALUE}, \texttt{INDEX}_i, \texttt{VALUE}_i)$ corresponds a 0-1 signature variable $\texttt{S}_i$ as well as the following signature constraint: $((\texttt{INDEX} = \texttt{INDEX}_i) \wedge (\texttt{VALUE} \leq \texttt{VALUE}_i)) \Leftrightarrow \texttt{S}_i$.

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).

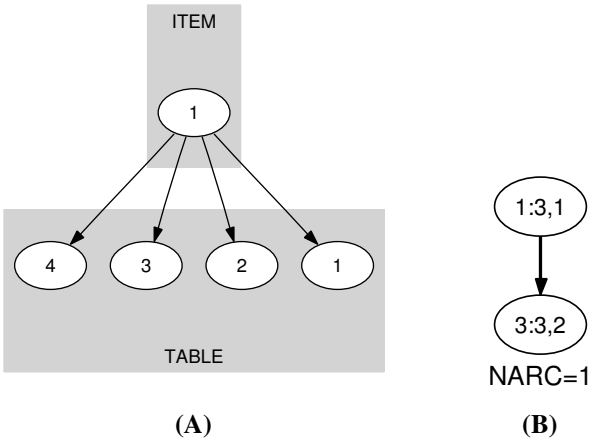


Figure 4.176: Initial and final graph of the element_lesseq constraint

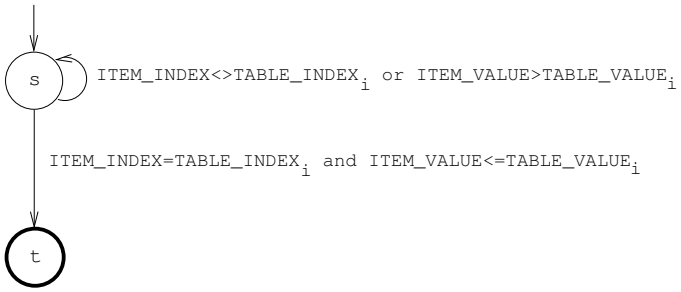


Figure 4.177: Automaton of the element_lesseq constraint

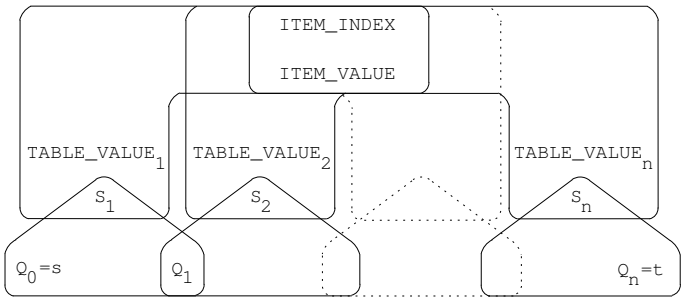


Figure 4.178: Hypergraph of the reformulation corresponding to the automaton of the element_lesseq constraint

4.84 element_matrix

Origin CHIP

Constraint element_matrix(MAX_I, MAX_J, INDEX_I, INDEX_J, MATRIX, VALUE)

Argument(s)

MAX_I	:	int
MAX_J	:	int
INDEX_I	:	dvar
INDEX_J	:	dvar
MATRIX	:	collection(i − int, j − int, v − int)
VALUE	:	dvar

Restriction(s)

MAX_I ≥ 1
MAX_J ≥ 1
INDEX_I ≥ 1
INDEX_I $\leq$ MAX_I
INDEX_J ≥ 1
INDEX_J $\leq$ MAX_J
required(MATRIX, [i, j, v])
increasing_seq(MATRIX, [i, j])
MATRIX.i ≥ 1
MATRIX.i $\leq$ MAX_I
MATRIX.j ≥ 1
MATRIX.j $\leq$ MAX_J
|MATRIX| = MAX_I $*$ MAX_J

Purpose

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.

Derived Collection(s)

$$\texttt{col}\left(\begin{array}{l}\texttt{ITEM} - \texttt{collection}(\texttt{index\_i} - \texttt{dvar}, \texttt{index\_j} - \texttt{dvar}, \texttt{value} - \texttt{dvar}),\\ [\texttt{item}(\texttt{index\_i} - \texttt{INDEX\_I}, \texttt{index\_j} - \texttt{INDEX\_J}, \texttt{value} - \texttt{VALUE})]\end{array}\right)$$

Arc input(s) ITEM MATRIX

Arc generator $PRODUCT \mapsto$ collection(item, matrix)

Arc arity 2

Arc constraint(s)

- item.index_i = matrix.i
- item.index_j = matrix.j
- item.value = matrix.v

Graph property(ies) **NARC** $= 1$

Example

$$\texttt{element\_matrix}\left(\begin{array}{l} 4,3,1,3,\left\{\begin{array}{lll} \texttt{i}-1 & \texttt{j}-1 & \texttt{v}-4,\\ \texttt{i}-1 & \texttt{j}-2 & \texttt{v}-1,\\ \texttt{i}-1 & \texttt{j}-3 & \texttt{v}-7,\\ \texttt{i}-2 & \texttt{j}-1 & \texttt{v}-1,\\ \texttt{i}-2 & \texttt{j}-2 & \texttt{v}-0,\\ \texttt{i}-2 & \texttt{j}-3 & \texttt{v}-8,\\ \texttt{i}-3 & \texttt{j}-1 & \texttt{v}-3,\\ \texttt{i}-3 & \texttt{j}-2 & \texttt{v}-2,\\ \texttt{i}-3 & \texttt{j}-3 & \texttt{v}-1,\\ \texttt{i}-4 & \texttt{j}-1 & \texttt{v}-0,\\ \texttt{i}-4 & \texttt{j}-2 & \texttt{v}-0,\\ \texttt{i}-4 & \texttt{j}-3 & \texttt{v}-6 \end{array}\right\},7 \end{array}\right)$$

Parts (A) and (B) of Figure 4.179 respectively show the initial and final graph. Since we use the $\mathbf{NARC}$ graph property, the unique arc of the final graph is stressed in bold.

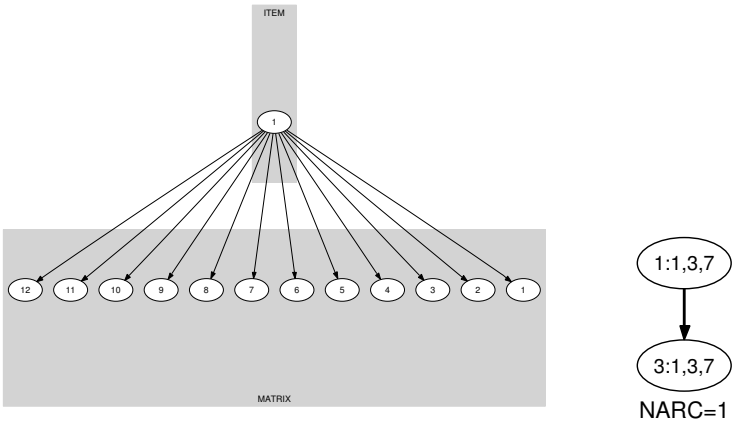


Figure 4.179: Initial and final graph of the `element_matrix` constraint

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 $\mathbf{NARC} = 1$ to $\mathbf{NARC} \geq 1$ and simplify $\underline{\overline{\mathbf{NARC}}}$ to $\overline{\mathbf{NARC}}$.

Automaton Figure 4.180 depicts the automaton associated to the `element_matrix` constraint. Let $\texttt{I}_k$, $\texttt{J}_k$ and $\texttt{V}_k$ respectively be the `i`, the `j` and the `v` k^{th} attributes of the `MATRIX` collection. To each sextuple $(\texttt{INDEX\_I}, \texttt{INDEX\_J}, \texttt{VALUE}, \texttt{I}_k, \texttt{J}_k, \texttt{V}_k)$ corresponds a 0-1 signature variable $\texttt{S}_k$ as well as the following signature constraint: $((\texttt{INDEX\_I} = \texttt{I}_k) \wedge (\texttt{INDEX\_J} = \texttt{J}_k) \wedge (\texttt{VALUE} = \texttt{V}_k)) \Leftrightarrow \texttt{S}_k$.

See also `element`.

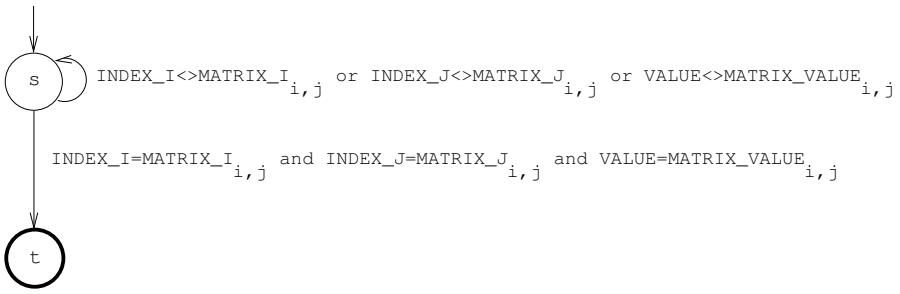


Figure 4.180: Automaton of the `element_matrix` constraint

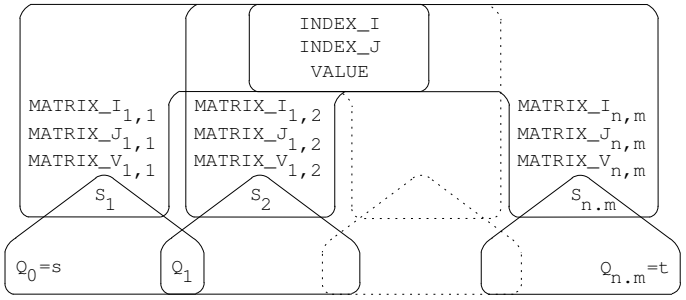


Figure 4.181: Hypergraph of the reformulation corresponding to the automaton of the `element_matrix` constraint

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(ITEM, TABLE, DEFAULT)

Usual name element

Argument(s)

ITEM	:	collection(index − dvar, value − dvar)
TABLE	:	collection(index − int, value − int)
DEFAULT	:	int

Restriction(s)

required(ITEM, [index, value])
ITEM.index ≥ 1
$|$ITEM$| = 1$
required(TABLE, [index, value])
TABLE.index ≥ 1
distinct(TABLE, index)

Purpose ITEM.value is equal to one of the entries of the table TABLE or to the default value DEFAULT if the entry ITEM.index does not exist in TABLE.

Derived Collection(s)

$$\text{col}(\text{DEF} - \text{collection}(\text{index} - \text{int}, \text{value} - \text{int}), [\text{item}(\text{index} - 0, \text{value} - \text{DEFAULT})])$$

$$\text{col}\left(\begin{array}{l}\text{TABLE\_DEF} - \text{collection}(\text{index} - \text{dvar}, \text{value} - \text{dvar}),\\ \left[\begin{array}{l}\text{item}(\text{index} - \text{TABLE.index}, \text{value} - \text{TABLE.value}),\\ \text{item}(\text{index} - \text{DEF.index}, \text{value} - \text{DEF.value})\end{array}\right]\end{array}\right)$$

Arc input(s) ITEM TABLE_DEF

Arc generator $PRODUCT \mapsto$ collection(item, table_def)

Arc arity 2

Arc constraint(s)

- item.value = table_def.value
- item.index = table_def.index $\vee$ table_def.index = 0

Graph property(ies) $\mathbf{NARC} \geq 1$

Example

$$\text{element\_sparse}\left(\begin{array}{l}\{\text{index} - 2\ \text{value} - 5\},\\ \left\{\begin{array}{ll}\text{index} - 1 & \text{value} - 6,\\ \text{index} - 2 & \text{value} - 5,\\ \text{index} - 4 & \text{value} - 2,\\ \text{index} - 8 & \text{value} - 9\end{array}\right\}, 5\end{array}\right)$$

Parts (A) and (B) of Figure 4.182 respectively show the initial and final graph. Since we use the $\mathbf{NARC}$ 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 DEFAULT occurs also in the table TABLE; 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 $\texttt{INDEX}_i$ and $\texttt{VALUE}_i$ respectively be the `index` and the `value` attributes of the i^{th} item of the `TABLE` collection. To each quintuple $(\texttt{INDEX}, \texttt{VALUE}, \texttt{DEFAULT}, \texttt{INDEX}_i, \texttt{VALUE}_i)$ corresponds a signature variable $\texttt{S}_i$ as well as the following signature constraint:

$$\begin{cases} (\texttt{INDEX} \neq \texttt{INDEX}_i \ \wedge \texttt{VALUE} \neq \texttt{DEFAULT}) & \Leftrightarrow \quad \texttt{S}_i = 0 \ \wedge \\ (\texttt{INDEX} = \texttt{INDEX}_i \ \wedge \texttt{VALUE} = \texttt{VALUE}_i \ ) & \Leftrightarrow \quad \texttt{S}_i = 1 \ \wedge \ . \\ (\texttt{INDEX} \neq \texttt{INDEX}_i \ \wedge \texttt{VALUE} = \texttt{DEFAULT}) & \Leftrightarrow \quad \texttt{S}_i = 2 \end{cases}$$

Usage

A sometimes more compact form of the `element` constraint: We are not obliged to specify explicitely 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.

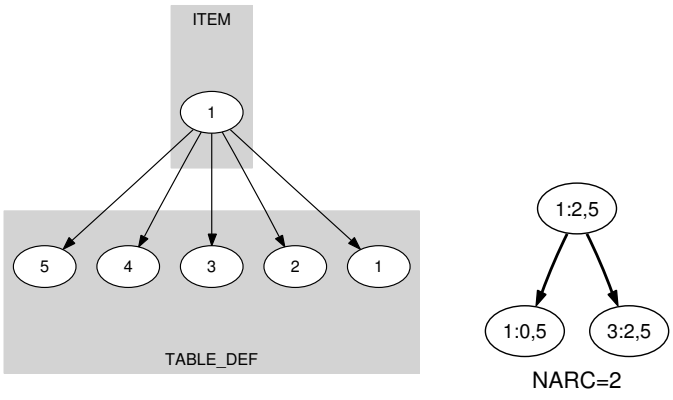


Figure 4.182: Initial and final graph of the element_sparse constraint

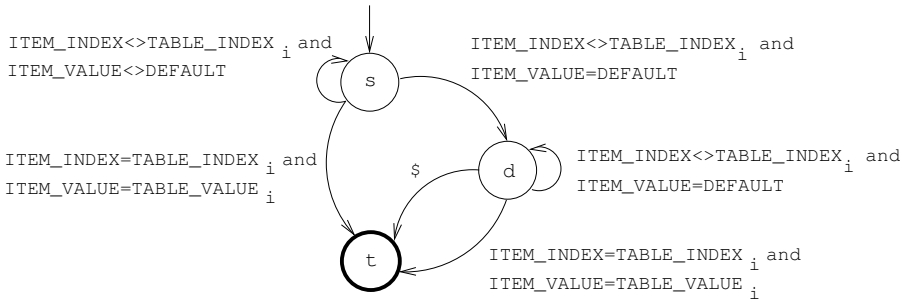


Figure 4.183: Automaton of the element_sparse constraint

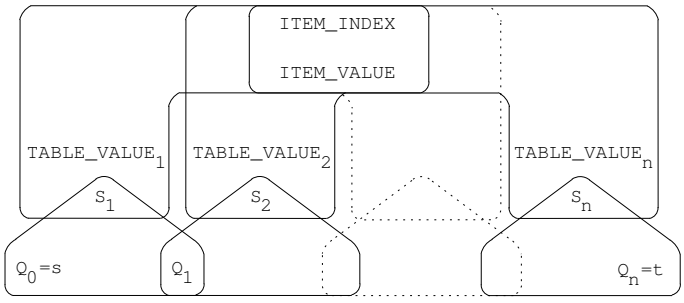


Figure 4.184: Hypergraph of the reformulation corresponding to the automaton of the element_sparse constraint

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 $\leq$ |TABLE|
required(TABLE, [index, value])
TABLE.index ≥ 1
TABLE.index $\leq$ |TABLE|
distinct(TABLE, index)

Purpose All the items of ITEMS should be equal to one of the entries of the table TABLE.

Arc input(s) ITEMS TABLE

Arc generator $PRODUCT \mapsto$ collection(items, table)

Arc arity 2

Arc constraint(s)
- items.index = table.index
- items.value = table.value

Graph property(ies) $\mathbf{NARC} =$ |ITEMS|

Example

$$\texttt{elements}\left(\begin{array}{l}\{\texttt{index}-4\ \texttt{value}-9, \texttt{index}-1\ \texttt{value}-6\},\\ \left\{\begin{array}{ll}\texttt{index}-1 & \texttt{value}-6,\\ \texttt{index}-2 & \texttt{value}-9,\\ \texttt{index}-3 & \texttt{value}-2,\\ \texttt{index}-4 & \texttt{value}-9\end{array}\right\}\end{array}\right)$$

Parts (A) and (B) of Figure 4.185 respectively show the initial and final graph. Since we use the $\mathbf{NARC}$ 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 $\mathbf{NARC} =$ |ITEMS| to $\mathbf{NARC} \geq$ |ITEMS|. So we can simplify $\underline{\overline{\mathbf{NARC}}}$ to $\overline{\mathbf{NARC}}$.

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.

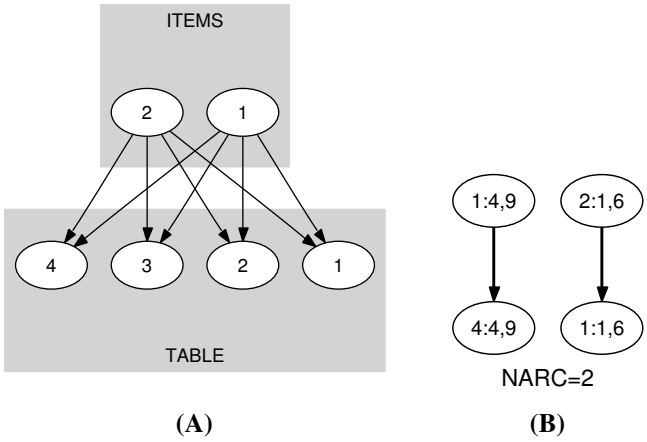


Figure 4.185: Initial and final graph of the `elements` constraint

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 $\leq$ |TABLE|
|ITEMS| = |TABLE|
required(TABLE, [index, value])
TABLE.index ≥ 1
TABLE.index $\leq$ |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.

Arc input(s) ITEMS TABLE

Arc generator $PRODUCT \mapsto$ collection(items, table)

Arc arity 2

Arc constraint(s)
- items.index = table.index
- items.value = table.value

Graph property(ies) **NVERTEX** = |ITEMS| + |TABLE|

Example

$$\texttt{elements\_alldifferent}\left(\left\langle \left\{\begin{array}{ll}\texttt{index}-2 & \texttt{value}-9,\\ \texttt{index}-1 & \texttt{value}-6,\\ \texttt{index}-4 & \texttt{value}-9,\\ \texttt{index}-3 & \texttt{value}-2\end{array}\right\}, \left\{\begin{array}{ll}\texttt{index}-1 & \texttt{value}-6,\\ \texttt{index}-2 & \texttt{value}-9,\\ \texttt{index}-3 & \texttt{value}-2,\\ \texttt{index}-4 & \texttt{value}-9\end{array}\right\}\right\rangle\right)$$

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.

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 $\mathbf{NVERTEX} = |\texttt{ITEMS}| + |\texttt{TABLE}|$ 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 $\texttt{items.index} = \texttt{table.index}$ of the arc constraint, and from the restriction $\texttt{distinct}(\texttt{TABLE.index})$ 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 $|\texttt{ITEMS}| + |\texttt{TABLE}|$ vertices one can simplify $\underline{\mathbf{NVERTEX}}$ to $\overline{\mathbf{NVERTEX}}$.

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:

$$\texttt{alldifferent}(\{\texttt{var}-2, \texttt{var}-1, \texttt{var}-4, \texttt{var}-3\})$$

$$\texttt{element}\left(\begin{array}{l}\{\ \texttt{index}-2 \quad \texttt{value}-9\ \},\\ \left\{\begin{array}{ll}\texttt{index}-1 & \texttt{value}-6,\\ \texttt{index}-2 & \texttt{value}-9,\\ \texttt{index}-3 & \texttt{value}-2,\\ \texttt{index}-4 & \texttt{value}-9\end{array}\right\}\end{array}\right)$$

$$\texttt{element}\left(\begin{array}{l}\{\ \texttt{index}-1 \quad \texttt{value}-6\ \},\\ \left\{\begin{array}{ll}\texttt{index}-1 & \texttt{value}-6,\\ \texttt{index}-2 & \texttt{value}-9,\\ \texttt{index}-3 & \texttt{value}-2,\\ \texttt{index}-4 & \texttt{value}-9\end{array}\right\}\end{array}\right)$$

$$\texttt{element}\left(\begin{array}{l}\{\ \texttt{index}-3 \quad \texttt{value}-2\ \},\\ \left\{\begin{array}{ll}\texttt{index}-1 & \texttt{value}-6,\\ \texttt{index}-2 & \texttt{value}-9,\\ \texttt{index}-3 & \texttt{value}-2,\\ \texttt{index}-4 & \texttt{value}-9\end{array}\right\}\end{array}\right)$$

$$\texttt{element}\left(\begin{array}{l}\{\ \texttt{index}-4 \quad \texttt{value}-9\ \},\\ \left\{\begin{array}{ll}\texttt{index}-1 & \texttt{value}-6,\\ \texttt{index}-2 & \texttt{value}-9,\\ \texttt{index}-3 & \texttt{value}-2,\\ \texttt{index}-4 & \texttt{value}-9\end{array}\right\}\end{array}\right)$$

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 $V_1V_2V_3V_4V_5V_6$ its expanded form.

The constraint:

$$\texttt{elements\_alldifferent}\left(\left\{\begin{array}{ll}\texttt{index}-V_1 & \texttt{value}-V_2,\\ \texttt{index}-V_2 & \texttt{value}-V_3,\\ \texttt{index}-V_3 & \texttt{value}-V_4,\\ \texttt{index}-V_4 & \texttt{value}-V_5,\\ \texttt{index}-V_5 & \texttt{value}-V_6,\\ \texttt{index}-V_6 & \texttt{value}-V_1\end{array}\right\},\left\{\begin{array}{ll}\texttt{index}-1 & \texttt{value}-S_1,\\ \texttt{index}-2 & \texttt{value}-S_2,\\ \texttt{index}-3 & \texttt{value}-S_3,\\ \texttt{index}-4 & \texttt{value}-S_4,\\ \texttt{index}-5 & \texttt{value}-S_5,\\ \texttt{index}-6 & \texttt{value}-S_6\end{array}\right\}\right)$$

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.

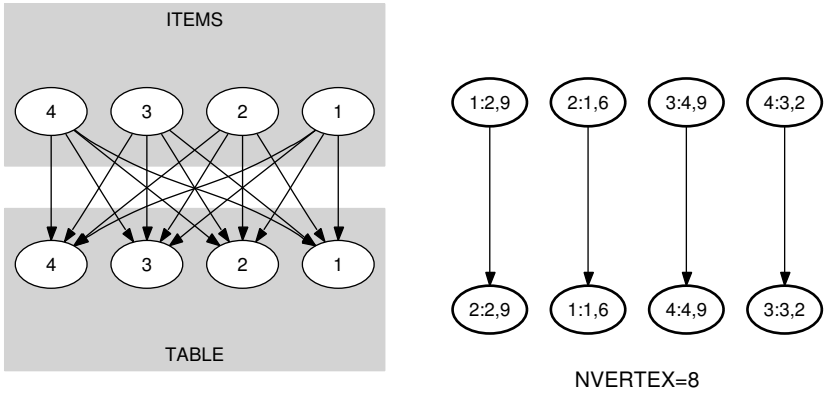


Figure 4.186: Initial and final graph of the `elements_alldifferent` constraint

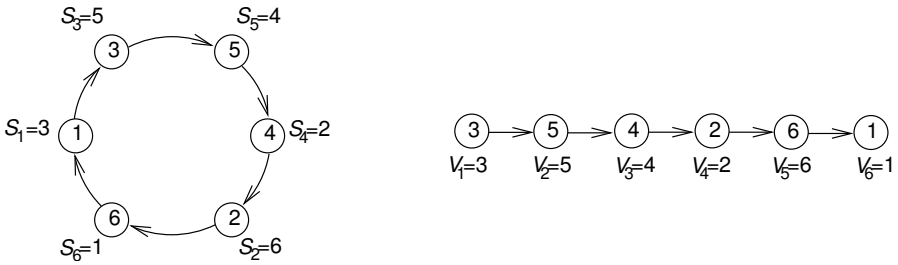


Figure 4.187: Two representations of a permutation containing one single cycle

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)

$$\texttt{col}(\texttt{DEF} - \texttt{collection}(\texttt{index} - \texttt{int}, \texttt{value} - \texttt{int}), [\texttt{item}(\texttt{index} - 0, \texttt{value} - \texttt{DEFAULT})])$$

$$\texttt{col}\left(\begin{array}{l}\texttt{TABLE\_DEF} - \texttt{collection}(\texttt{index} - \texttt{dvar}, \texttt{value} - \texttt{dvar}),\\ \left[\begin{array}{l}\texttt{item}(\texttt{index} - \texttt{TABLE.index}, \texttt{value} - \texttt{TABLE.index}),\\ \texttt{item}(\texttt{index} - \texttt{DEF.index}, \texttt{value} - \texttt{DEF.value})\end{array}\right]\end{array}\right)$$

Arc input(s) ITEMS TABLE_DEF

Arc generator $PRODUCT \mapsto \texttt{collection}(\texttt{items}, \texttt{table\_def})$

Arc arity 2

Arc constraint(s)

- items.value = table_def.value
- items.index = table_def.index ∨ table_def.index = 0

Graph property(ies) **NSOURCE** = |ITEMS|

Example

$$\texttt{elements\_sparse}\left(\begin{array}{l}\left\{\begin{array}{ll}\texttt{index} - 8 & \texttt{value} - 9,\\ \texttt{index} - 3 & \texttt{value} - 5,\\ \texttt{index} - 2 & \texttt{value} - 5\end{array}\right\},\\ \left\{\begin{array}{ll}\texttt{index} - 1 & \texttt{value} - 6,\\ \texttt{index} - 2 & \texttt{value} - 5,\\ \texttt{index} - 4 & \texttt{value} - 2,\\ \texttt{index} - 8 & \texttt{value} - 9\end{array}\right\}, 5\end{array}\right)$$

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.

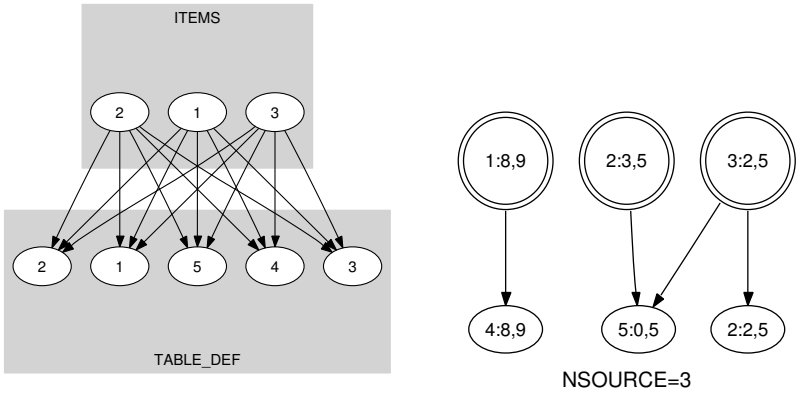
Graph model An item of the ITEMS collection may have up to two successors (see for instance the third item of the ITEMS collection of the previous example). Therefore we use the graph property $\mathbf{NSOURCE} = |\texttt{ITEMS}|$ for enforcing the fact that each item of the ITEMS collection has at least 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 $\mathbf{NSOURCE} = |\texttt{ITEMS}|$ to $\mathbf{NSOURCE} \geq |\texttt{ITEMS}|$ and simplify $\overline{\mathbf{NSOURCE}}$ to $\overline{\mathbf{NSOURCE}}$.

Usage Used for replacing several element constraints sharing exactly the same sparse table by one single constraint.

See also element, element_sparse.

Key words data constraint, table, shared table, sparse table, sparse functional dependency, derived collection.



(A) **(B)**

Figure 4.188: Initial and final graph of the `elements_sparse` constraint

4.89 eq_set

Origin Used for defining alldifferent_between_sets.

Constraint eq_set(SET1, SET2)

Argument(s)

SET1	:	svar
SET2	:	svar

Purpose Constraint the set SET1 to be equal to the set SET2.

Example eq_set($\{3, 5\}, \{3, 5\}$)

Used in alldifferent_between_sets.

Key words predefined constraint, binary constraint, equality, constraint involving set variables.

4.90 exactly

Origin Derived from `atleast` and `atmost`.

Constraint `exactly`(`N`, `VARIABLES`, `VALUE`)

Argument(s)

`N`	:	`int`
`VARIABLES`	:	`collection`(`var` − `dvar`)
`VALUE`	:	`int`

Restriction(s) $\texttt{N} \geq 0$
$\texttt{N} \leq |\texttt{VARIABLES}|$
`required`(`VARIABLES`, `var`)

Purpose Exactly `N` variables of the `VARIABLES` collection are assigned to value `VALUE`.

Arc input(s) `VARIABLES`

Arc generator $SELF \mapsto$ `collection`(`variables`)

Arc arity 1

Arc constraint(s) `variables.var` = `VALUE`

Graph property(ies) $\mathbf{NARC} = \texttt{N}$

Example `exactly`$(2, \{\texttt{var} - 4, \texttt{var} - 2, \texttt{var} - 4, \texttt{var} - 5\}, 4)$

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.

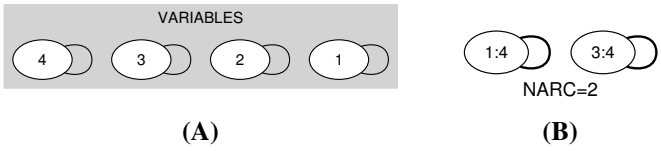


Figure 4.189: Initial and final graph of the `exactly` constraint

Graph model Since we use a unary arc constraint (`VALUE` is fixed) we employ the $SELF$ 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 `exactly` constraint. To each variable $\texttt{VAR}_i$ of the collection `VARIABLES` corresponds a 0-1 signature variable $\texttt{S}_i$. The following signature constraint links $\texttt{VAR}_i$ and $\texttt{S}_i$: $\texttt{VAR}_i = \texttt{VALUE} \Leftrightarrow \texttt{S}_i$.

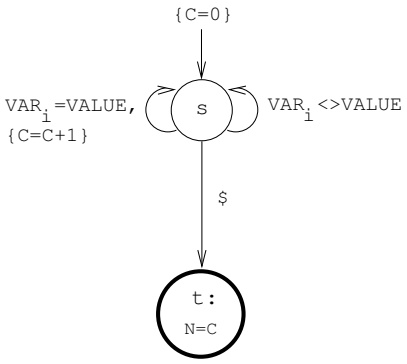


Figure 4.190: Automaton of the `exactly` constraint

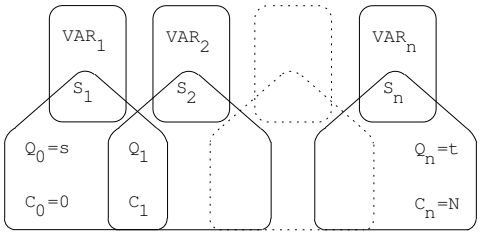


Figure 4.191: Hypergraph of the reformulation corresponding to the automaton of the `exactly` constraint

See also `atleast`, `atmost`, `among`.

Key words value constraint, counting constraint, automaton, automaton with counters, alpha-acyclic constraint network(2).

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 ≥ 0
VALUES.noccurrence $\leq$ |VARIABLES|

Purpose

Each value VALUES$[i]$.val $(1 \leq i \leq |\text{VALUES}|)$ should be taken by exactly VALUES$[i]$.noccurrence variables of the VARIABLES collection.

For all items of VALUES:

Arc input(s) VARIABLES

Arc generator $SELF \mapsto$ collection(variables)

Arc arity 1

Arc constraint(s) variables.var = VALUES.val

Graph property(ies) $\mathbf{NVERTEX}$ = VALUES.noccurrence

Example

$$\text{global\_cardinality}\left(\left\{\begin{array}{l}\text{var}-3,\\ \text{var}-3,\\ \text{var}-8,\\ \text{var}-6\end{array}\right\},\left\{\begin{array}{ll}\text{val}-3 & \text{noccurrence}-2,\\ \text{val}-5 & \text{noccurrence}-0,\\ \text{val}-6 & \text{noccurrence}-1\end{array}\right\}\right)$$

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 $\mathbf{NVERTEX}$ 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 $\mathtt{S}_i$, which is equal to 0. To each item of the collection VALUES corresponds a signature variable $\mathtt{S}_{i+|\mathtt{VARIABLES}|}$, 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 $S = (s_0, s_1, \ldots, s_n)$ 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:

$$\mathtt{global\_cardinality}\left(\begin{array}{l}\left\{\ \mathtt{var}-s_0, \mathtt{var}-s_1, \ldots, \mathtt{var}-s_n\ \right\},\\ \left\{\begin{array}{ll}\mathtt{val}-0 & \mathtt{noccurrence}-s_0,\\ \mathtt{val}-1 & \mathtt{noccurrence}-s_1,\\ & \vdots\\ \mathtt{val}-n & \mathtt{noccurrence}-s_n\end{array}\right\}\end{array}\right)$$

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.

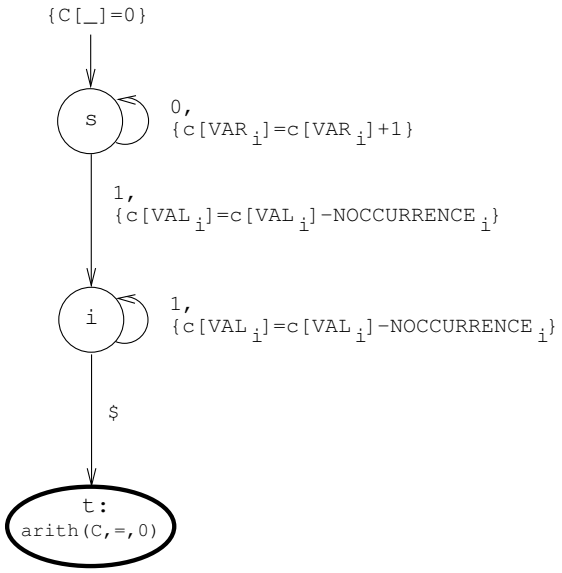


Figure 4.193: Automaton of the `global_cardinality` constraint

4.92 global_cardinality_low_up

Origin Used for defining sliding_distribution.

Constraint global_cardinality_low_up(VARIABLES, VALUES)

Argument(s)

VARIABLES	:	collection(var − dvar)
VALUES	:	collection(val − int, omin − int, omax − int)

Restriction(s)

required(VARIABLES, var)
|VALUES| > 0
required(VALUES, [val, omin, omax])
distinct(VALUES, val)
VALUES.omin ≥ 0
VALUES.omax ≤ |VARIABLES|
VALUES.omin ≤ VALUES.omax

Purpose Each value VALUES[i].val ($1 \leq i \leq |\text{VALUES}|$) should be taken by at least VALUES[i].omin and at most VALUES[i].omax variables of the VARIABLES collection.

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.omin
- **NVERTEX** ≤ VALUES.omax

Example

$$\text{global\_cardinality\_low\_up}\left(\left\{\begin{array}{l}\text{var}-3,\\ \text{var}-3,\\ \text{var}-8,\\ \text{var}-6\end{array}\right\},\left\{\begin{array}{lll}\text{val}-3 & \text{omin}-2 & \text{omax}-3,\\ \text{val}-5 & \text{omin}-0 & \text{omax}-1,\\ \text{val}-6 & \text{omin}-1 & \text{omax}-2\end{array}\right\}\right)$$

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 `sliding_distribution`.

See also `global_cardinality`, `sliding_distribution`.

Key words value constraint, assignment, flow.

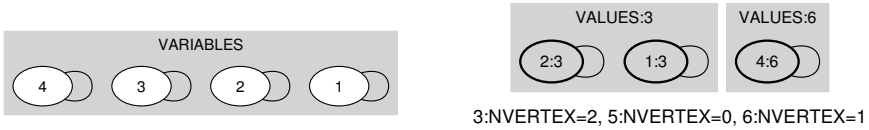


Figure 4.194: Initial and final graph of the `global_cardinality_low_up` constraint

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 $SELF \mapsto$ collection(variables)

Arc arity 1

Arc constraint(s) variables.var = VALUES.val

Graph property(ies) **NVERTEX** = VALUES.noccurrence

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) **SUM_WEIGHT_ARC**(MATRIX[(variables.key − 1) ∗ |VALUES| + values.key].c) = COST

Example

$$\texttt{global\_cardinality\_with\_costs}\left(\left\{\begin{array}{l}\texttt{var}-3,\\ \texttt{var}-3,\\ \texttt{var}-3,\\ \texttt{var}-6\end{array}\right\},\ \left\{\begin{array}{ll}\texttt{val}-3 & \texttt{noccurrence}-3,\\ \texttt{val}-5 & \texttt{noccurrence}-0,\\ \texttt{val}-6 & \texttt{noccurrence}-1\end{array}\right\},\ \left\{\begin{array}{lll}\texttt{i}-1 & \texttt{j}-1 & \texttt{c}-4,\\ \texttt{i}-1 & \texttt{j}-2 & \texttt{c}-1,\\ \texttt{i}-1 & \texttt{j}-3 & \texttt{c}-7,\\ \texttt{i}-2 & \texttt{j}-1 & \texttt{c}-1,\\ \texttt{i}-2 & \texttt{j}-2 & \texttt{c}-0,\\ \texttt{i}-2 & \texttt{j}-3 & \texttt{c}-8,\\ \texttt{i}-3 & \texttt{j}-1 & \texttt{c}-3,\\ \texttt{i}-3 & \texttt{j}-2 & \texttt{c}-2,\\ \texttt{i}-3 & \texttt{j}-3 & \texttt{c}-1,\\ \texttt{i}-4 & \texttt{j}-1 & \texttt{c}-0,\\ \texttt{i}-4 & \texttt{j}-2 & \texttt{c}-0,\\ \texttt{i}-4 & \texttt{j}-3 & \texttt{c}-6\end{array}\right\},14\right)$$

Parts (A) and (B) of Figure 4.195 respectively show the initial and final graph associated to the second graph constraint.

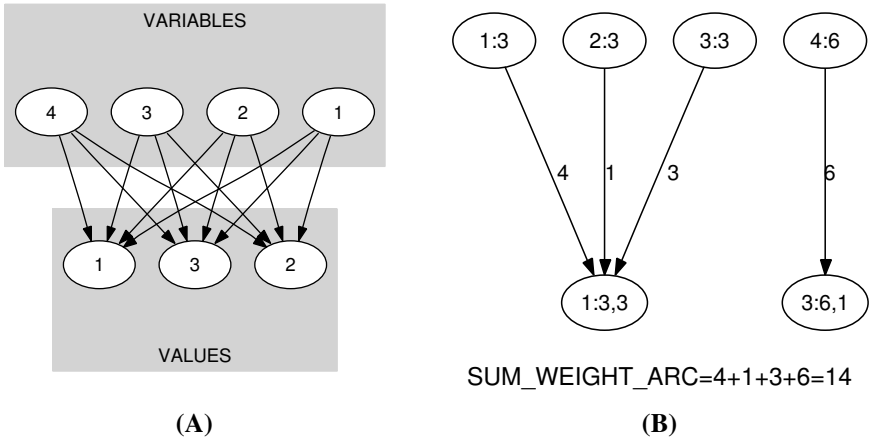


Figure 4.195: Initial and final graph of the global_cardinality_with_costs constraint

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 $((i-1)\cdot|\texttt{VALUES}|+j)^{th}$ 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.

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 $\texttt{alldifferent}(\texttt{X}_1,\texttt{X}_2,\ldots,\texttt{X}_\texttt{n})$ and $\alpha_1\cdot\texttt{X}_1+\alpha_2\cdot\texttt{X}_2+\ldots+\alpha_n\cdot\texttt{X}_\texttt{n}=\texttt{COST}$. For this purpose we set the domain of the noccurrence variables to $\{0,1\}$ and the cost attribute c of a variable $\texttt{X}_\texttt{i}$ and one of its potential value j to $\alpha_i\cdot\texttt{j}$. 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.

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.

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** ≤ 1

Example global_contiguity $\left(\left\{ \begin{array}{l} \text{var} - 0, \\ \text{var} - 1, \\ \text{var} - 1, \\ \text{var} - 0 \end{array} \right\} \right)$

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.

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.

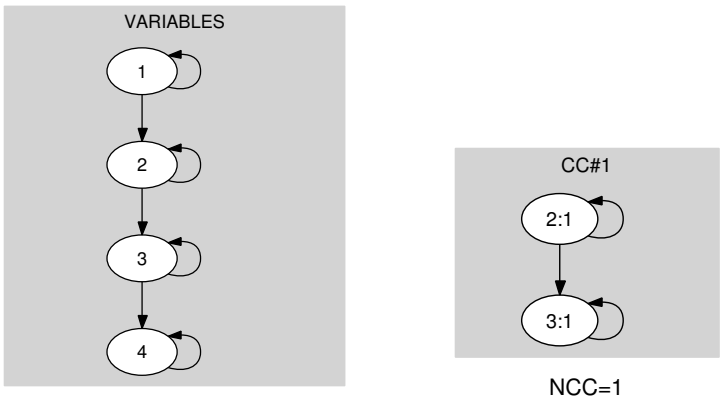


Figure 4.196: Initial and final graph of the `global_contiguity` constraint

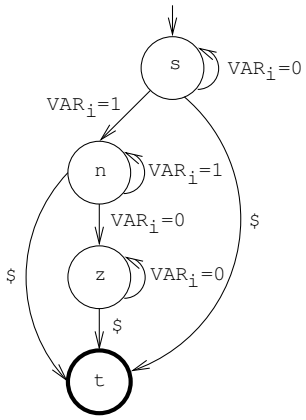


Figure 4.197: Automaton of the `global_contiguity` constraint

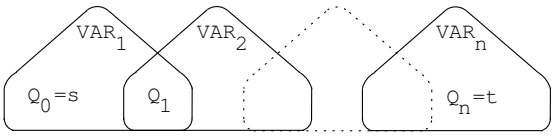


Figure 4.198: Hypergraph of the reformulation corresponding to the automaton of the `global_contiguity` constraint

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(PAIRS − collection(x − dvar, y − dvar), [> −item(x − VARIABLES.var, y − VARIABLES.var)])

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})

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.

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) $a_1 < \cdots < a_k$ 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].

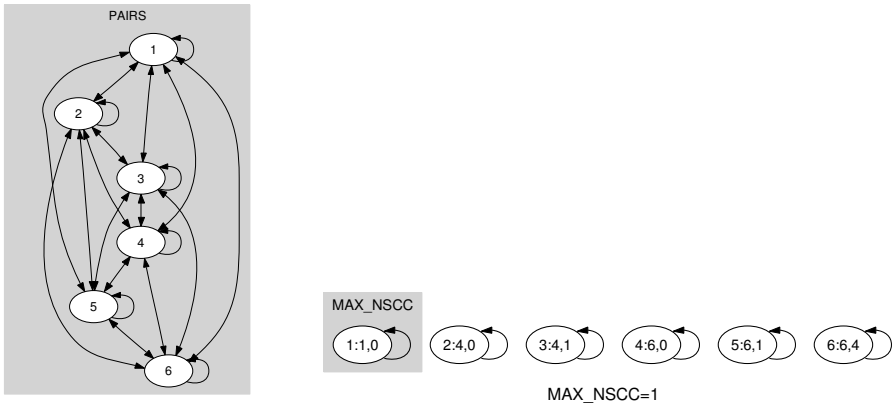


Figure 4.199: Initial and final graph of the `golomb` constraint

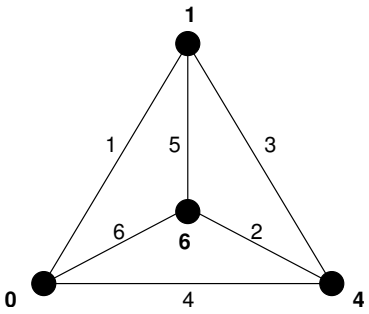


Figure 4.200: Graphical representation of the solution 0,1,4,6

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 $\leq |\text{NODES}|$

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 $CLIQUE(<) \mapsto$ collection(n1, n2)

Arc arity 2

Arc constraint(s)

- 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)
- $\begin{array}{l}(\text{n2.x} - \text{NODES[n1.succ].x}) * (\text{NODES[n1.succ].y} - \text{n1.y}) - \\ (\text{NODES[n1.succ].x} - \text{n1.x}) * (\text{n2.y} - \text{NODES[n1.succ].y})\end{array} \neq 0$
- $\begin{array}{l}(\text{NODES[n2.succ].x} - \text{NODES[n1.succ].x}) * (\text{n2.y} - \text{n1.y}) - \\ (\text{n2.x} - \text{n1.x}) * (\text{NODES[n2.succ].y} - \text{NODES[n1.succ].y})\end{array} \neq 0$
- $\text{sign}\left(\begin{array}{l}(\text{n2.x} - \text{NODES[n1.succ].x}) * (\text{NODES[n1.succ].y} - \text{n1.y}) - \\ (\text{NODES[n1.succ].x} - \text{n1.x}) * (\text{n2.y} - \text{NODES[n1.succ].y})\end{array}\right) \neq$ $\text{sign}\left(\begin{array}{l}(\text{NODES[n2.succ].x} - \text{NODES[n1.succ].x}) * (\text{n2.y} - \text{n1.y}) - \\ (\text{n2.x} - \text{n1.x}) * (\text{NODES[n2.succ].y} - \text{NODES[n1.succ].y})\end{array}\right)$

Graph property(ies) $\mathbf{NARC} = \text{NCROSS}$

Example

$$\text{graph\_crossing}\left(2, \left\{\begin{array}{lll} \text{succ}-1 & \text{x}-4 & \text{y}-7, \\ \text{succ}-1 & \text{x}-2 & \text{y}-5, \\ \text{succ}-1 & \text{x}-7 & \text{y}-6, \\ \text{succ}-2 & \text{x}-1 & \text{y}-2, \\ \text{succ}-3 & \text{x}-2 & \text{y}-2, \\ \text{succ}-2 & \text{x}-5 & \text{y}-3, \\ \text{succ}-3 & \text{x}-8 & \text{y}-2, \\ \text{succ}-9 & \text{x}-6 & \text{y}-2, \\ \text{succ}-10 & \text{x}-10 & \text{y}-6, \\ \text{succ}-8 & \text{x}-10 & \text{y}-1 \end{array}\right\}\right)$$

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.

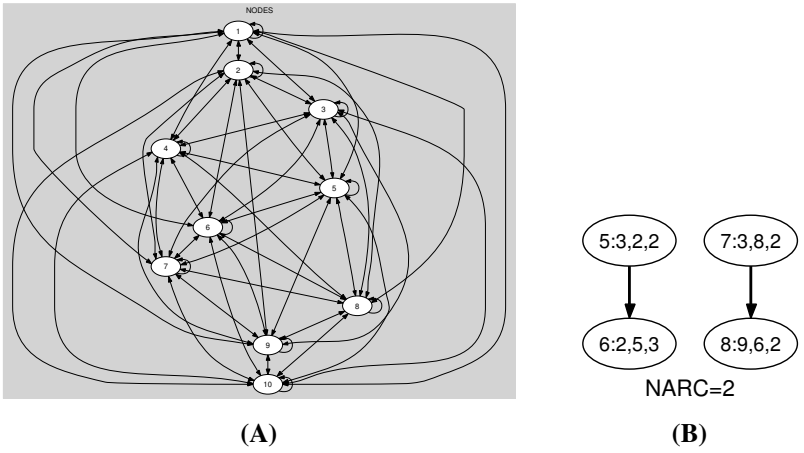


Figure 4.201: Initial and final graph of the graph_crossing constraint

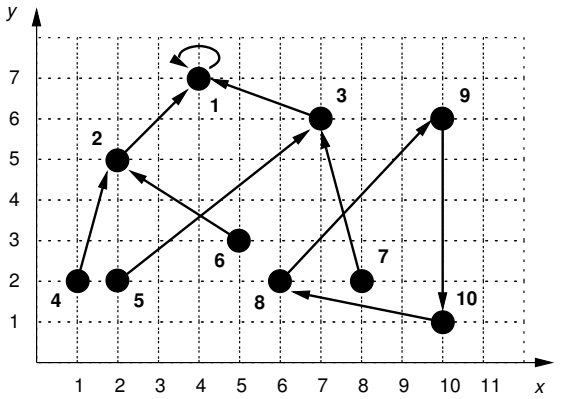


Figure 4.202: A graph covering with 2 line-segments intersections

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.

4.97 group

Origin CHIP

Constraint group(NGROUP, MIN_SIZE, MAX_SIZE, MIN_DIST, MAX_DIST, NVAL, VARIABLES, VALUES)

Argument(s)

NGROUP	:	dvar
MIN_SIZE	:	dvar
MAX_SIZE	:	dvar
MIN_DIST	:	dvar
MAX_DIST	:	dvar
NVAL	:	dvar
VARIABLES	:	collection(var − dvar)
VALUES	:	collection(val − int)

Restriction(s)

NGROUP ≥ 0
MIN_SIZE ≥ 0
MAX_SIZE $\geq$ MIN_SIZE
MIN_DIST ≥ 0
MAX_DIST $\geq$ MIN_DIST
NVAL ≥ 0
required(VARIABLES, var)
required(VALUES, val)
distinct(VALUES, val)

Purpose

Let n be the number of variables of the collection VARIABLES. Let $X_i, X_{i+1}, \ldots, X_j$ $(1 \leq i \leq j \leq n)$ be consecutive variables of the collection of variables VARIABLES such that all the following conditions simultaneously apply:

- All variables $X_i, \ldots, X_j$ 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 consecutives groups or between one border and one group,
- MAX_DIST is the maximum number of variables between two consecutives 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

$PATH \mapsto$ collection(variables1, variables2)
$LOOP \mapsto$ collection(variables1, variables2)

Arc arity 2

Arc constraint(s)
- in(variables1.var, VALUES)
- in(variables2.var, VALUES)

Graph property(ies)
- **NCC** = NGROUP
- **MIN_NCC** = MIN_SIZE
- **MAX_NCC** = MAX_SIZE
- **NVERTEX** = NVAL

Arc input(s) VARIABLES

Arc generator $PATH \mapsto$ collection(variables1, variables2)
$LOOP \mapsto$ collection(variables1, variables2)

Arc arity 2

Arc constraint(s)
- not_in(variables1.var, VALUES)
- not_in(variables2.var, VALUES)

Graph property(ies)
- **MIN_NCC** = MIN_DIST
- **MAX_NCC** = MAX_DIST

Example

$$\texttt{group}\left(\begin{array}{l} 2,1,2,2,4,3,\left\{\begin{array}{l}\texttt{var}-2,\\ \texttt{var}-8,\\ \texttt{var}-1,\\ \texttt{var}-7,\\ \texttt{var}-4,\\ \texttt{var}-5,\\ \texttt{var}-1,\\ \texttt{var}-1,\\ \texttt{var}-1\end{array}\right\},\\ \left\{\begin{array}{l}\texttt{val}-0,\\ \texttt{val}-2,\\ \texttt{val}-4,\\ \texttt{val}-6,\\ \texttt{val}-8\end{array}\right\}\end{array}\right)$$

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 $\mathbf{NVERTEX}$ graph property, the vertices of the final graph are stressed in bold. In addition, since we use the $\mathbf{MIN\_NCC}$ and the $\mathbf{MAX\_NCC}$ graph properties, we also show the smallest and largest connected components of the final graph.

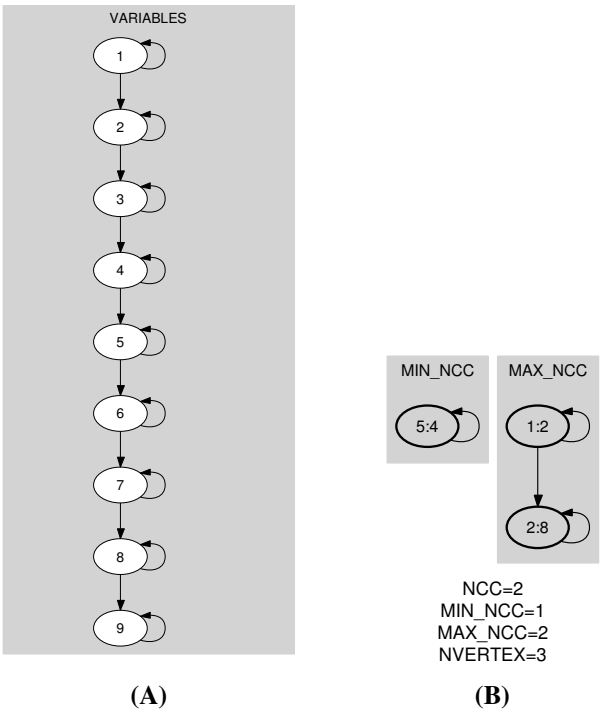


Figure 4.203: Initial and final graph of the group constraint

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 variables1.var $\in$ VALUES $\wedge$ variables2.var $\in$ VALUES.

This produces an initial graph depicted in part (A) of Figure 4.203. We use $PATH$ $LOOP$ and the binary constraint variables1.var $\in$ VALUES $\wedge$ variables2.var $\in$ 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 $\mathtt{S}_i$ for each variable $\mathtt{VAR}_i$ of the collection VARIABLES. The following signature constraint links $\mathtt{VAR}_i$ and $\mathtt{S}_i$: $\mathtt{VAR}_i \in \mathtt{VALUES} \Leftrightarrow \mathtt{S}_i$.

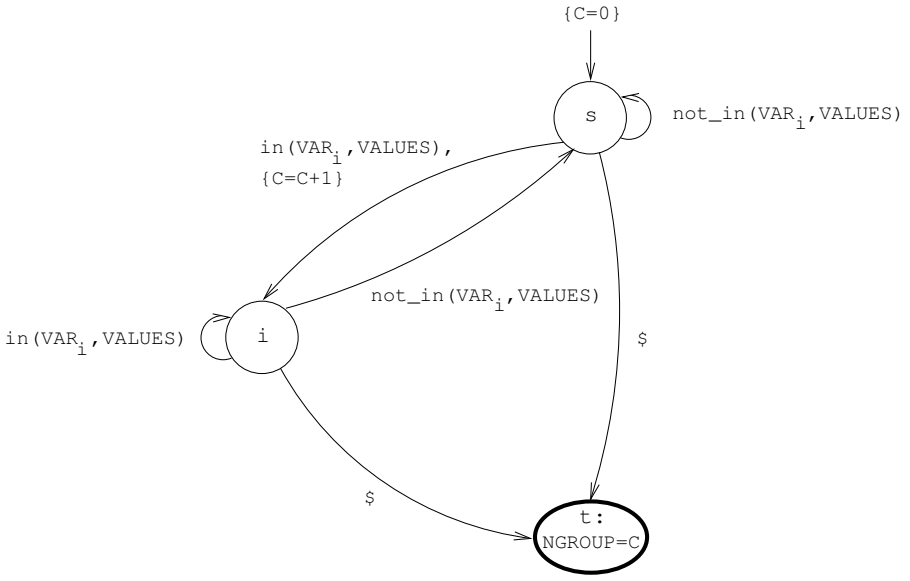


Figure 4.204: Automaton for the NGROUP parameter of the group constraint

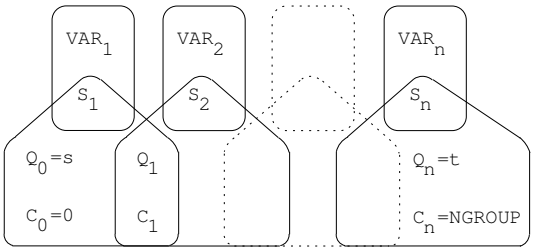


Figure 4.205: Hypergraph of the reformulation corresponding to the automaton of the NGROUP parameter of the group constraint

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 person on day i. A complete period of

520 $\overline{\textbf{MAX\_NCC}}$, $\overline{\textbf{MIN\_NCC}}$, $\overline{\textbf{NCC}}$, $\overline{\textbf{NVERTEX}}$, $PATH$, $LOOP$; $\overline{\textbf{MAX\_NCC}}$, $\overline{\textbf{MIN\_NCC}}$, $PATH$, $LOOP$

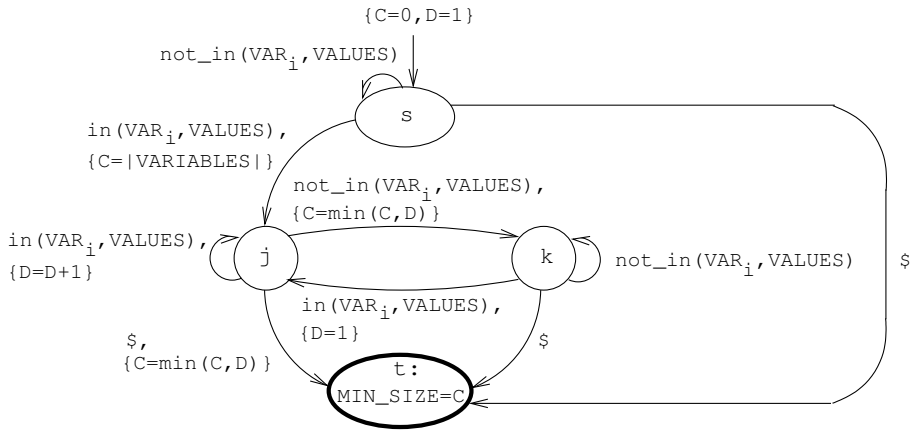


Figure 4.206: Automaton for the MIN_SIZE parameter of the group constraint

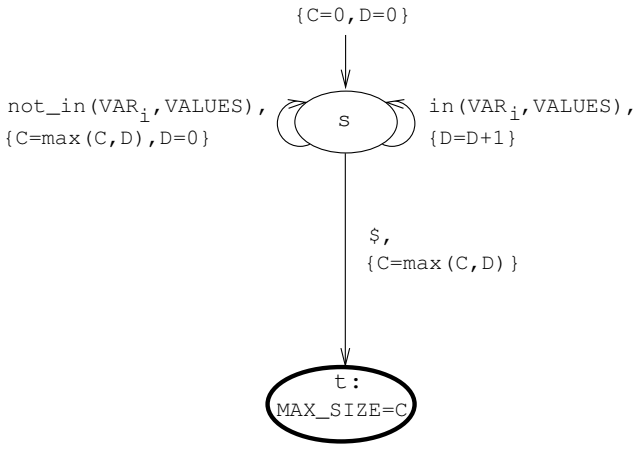


Figure 4.207: Automaton for the MAX_SIZE parameter of the group constraint

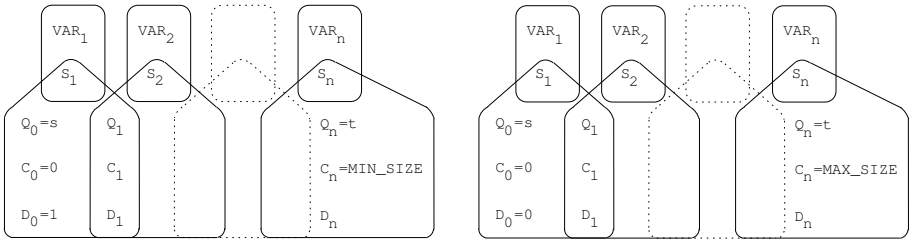


Figure 4.208: Hypergraphs of the reformulations corresponding to the automata of the MIN_SIZE and MAX_SIZE parameters of the group constraint

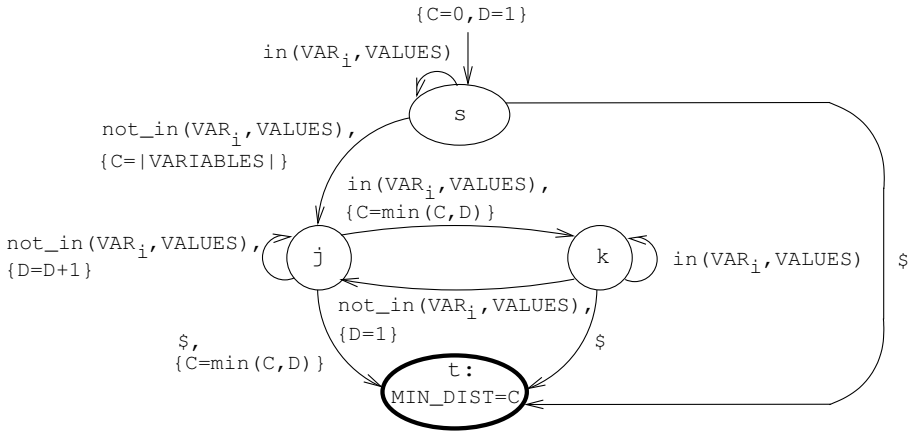


Figure 4.209: Automaton for the MIN_DIST parameter of the group constraint

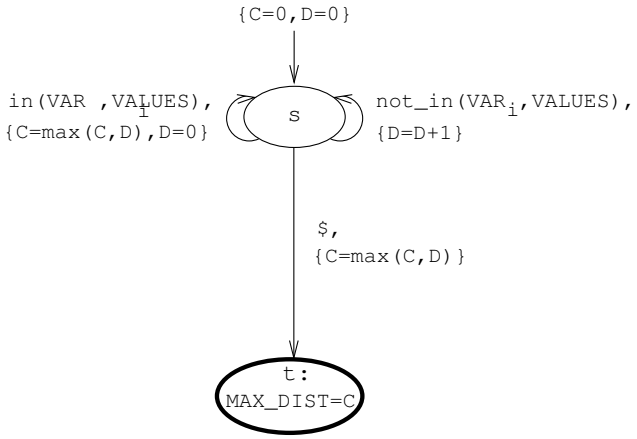


Figure 4.210: Automaton for the MAX_DIST parameter of the group constraint

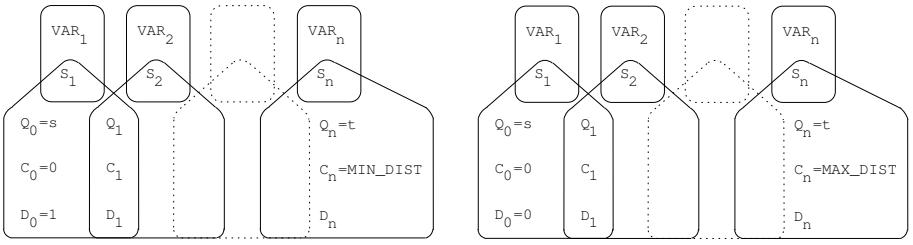


Figure 4.211: Hypergraphs of the reformulations corresponding to the automata of the MIN_DIST and MAX_DIST parameters of the group constraint

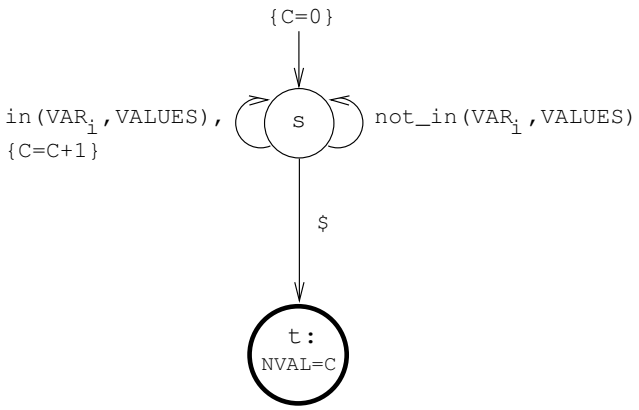


Figure 4.212: Automaton for the NVAL parameter of the group constraint

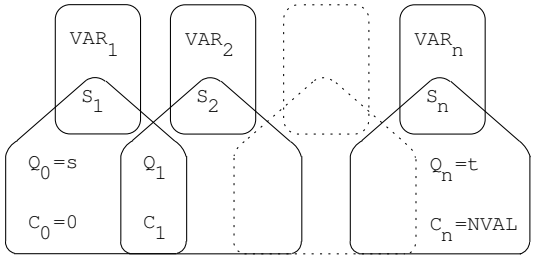


Figure 4.213: Hypergraph of the reformulation corresponding to the automaton of the NVAL parameter of the group constraint

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 group.

Constraint group_skip_isolated_item(NGROUP, MIN_SIZE, MAX_SIZE, NVAL, VARIABLES, VALUES)

Argument(s)

NGROUP	:	dvar
MIN_SIZE	:	dvar
MAX_SIZE	:	dvar
NVAL	:	dvar
VARIABLES	:	collection(var − dvar)
VALUES	:	collection(val − int)

Restriction(s)

NGROUP ≥ 0
MIN_SIZE ≥ 0
MAX_SIZE $\geq$ MIN_SIZE
NVAL ≥ 0
required(VARIABLES, var)
required(VALUES, val)
distinct(VALUES, val)

Purpose

Let n be the number of variables of the collection VARIABLES. Let $X_i, X_{i+1}, \ldots, X_j$ $(1 \leq i < j \leq n)$ be consecutive variables of the collection of variables VARIABLES such that the following conditions apply:

- All variables $X_i, \ldots, X_j$ 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) VARIABLES

Arc generator $CHAIN \mapsto$ collection(variables1, variables2)

Arc arity 2

Arc constraint(s)

- in(variables1.var, VALUES)
- in(variables2.var, VALUES)

Graph property(ies)

- **NSCC** = NGROUP
- **MIN_NSCC** = MIN_SIZE
- **MAX_NSCC** = MAX_SIZE
- **NVERTEX** = NVAL

Example

$$\texttt{group\_skip\_isolated\_item}\left(\begin{array}{l} 1,2,2,3,\left\{\begin{array}{l}\texttt{var}-2,\\ \texttt{var}-8,\\ \texttt{var}-1,\\ \texttt{var}-7,\\ \texttt{var}-4,\\ \texttt{var}-5,\\ \texttt{var}-1,\\ \texttt{var}-1,\\ \texttt{var}-1\end{array}\right),\\ \left\{\begin{array}{l}\texttt{val}-0,\\ \texttt{val}-2,\\ \texttt{val}-4,\\ \texttt{val}-6,\\ \texttt{val}-8\end{array}\right\}\end{array}\right)$$

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 variables1.var $\in$ VALUES $\wedge$ variables2.var $\in$ 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 $\texttt{S}_i$ for each variable $\texttt{VAR}_i$ of the collection VARIABLES. The following signature constraint links $\texttt{VAR}_i$ and $\texttt{S}_i$: $\texttt{VAR}_i \in \texttt{VALUES} \Leftrightarrow \texttt{S}_i$.

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.

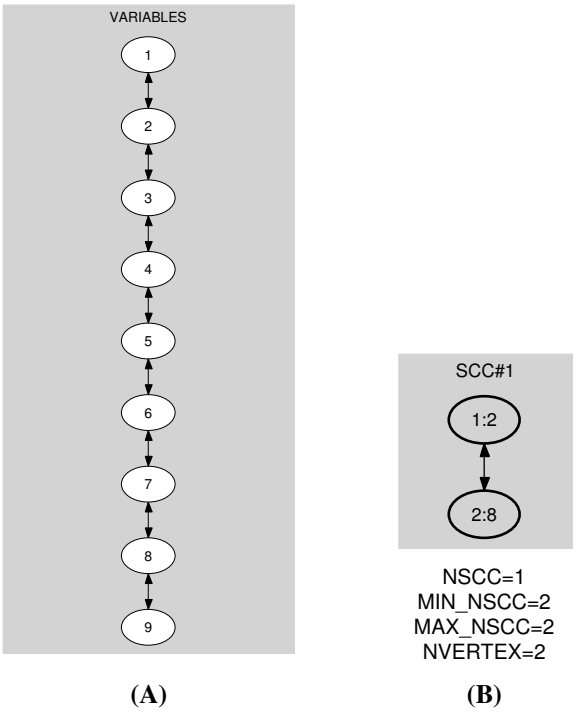


Figure 4.214: Initial and final graph of the `group_skip_isolated_item` constraint

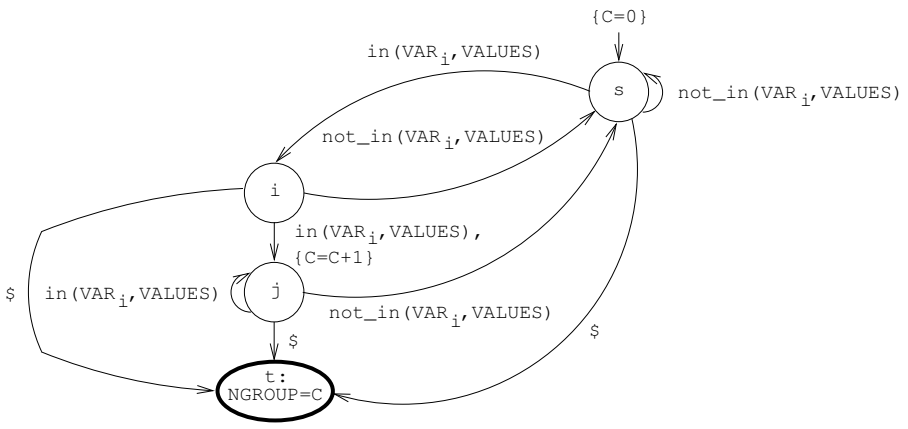


Figure 4.215: Automaton for the `NGROUP` parameter of the `group_skip_isolated_item` constraint

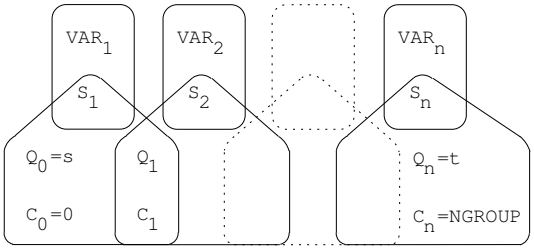


Figure 4.216: Hypergraph of the reformulation corresponding to the automaton of the NGROUP parameter of the group_skip_isolated_item constraint

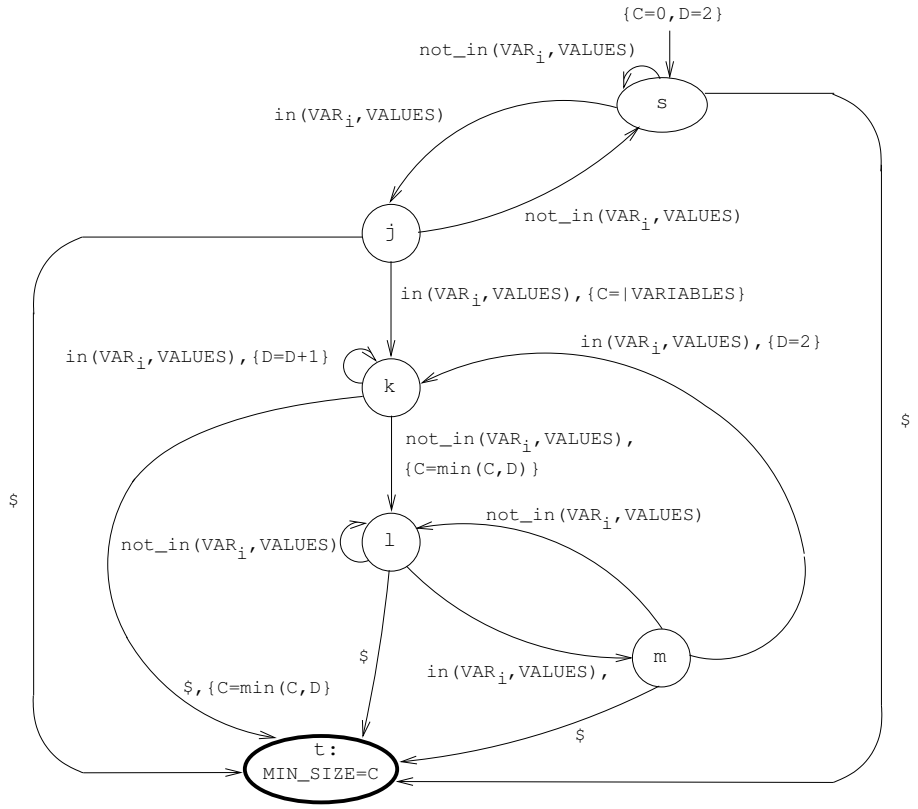


Figure 4.217: Automaton for the MIN_SIZE parameter of the group_skip_isolated_item constraint

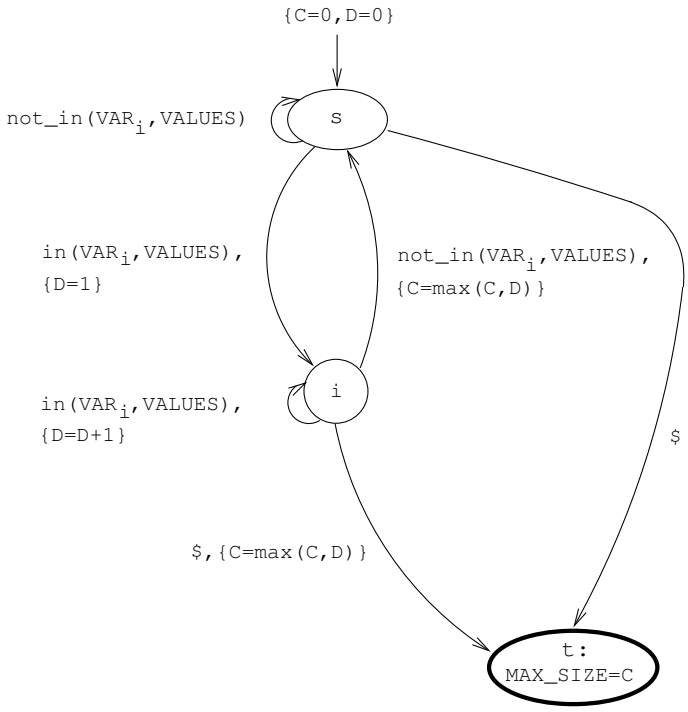


Figure 4.218: Automaton for the MAX_SIZE parameter of the group_skip_isolated_item constraint

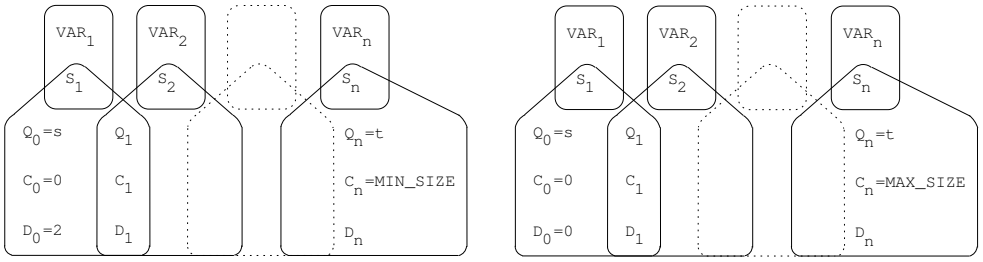


Figure 4.219: Hypergraphs of the reformulations corresponding to the automata of the MIN_SIZE and MAX_SIZE parameters of the group_skip_isolated_item constraint

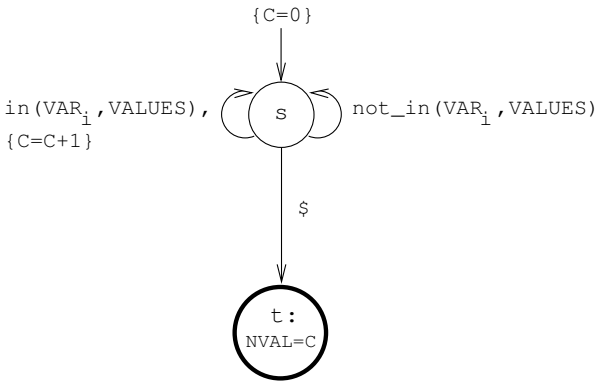


Figure 4.220: Automaton for the NVAL parameter of the group_skip_isolated_item constraint

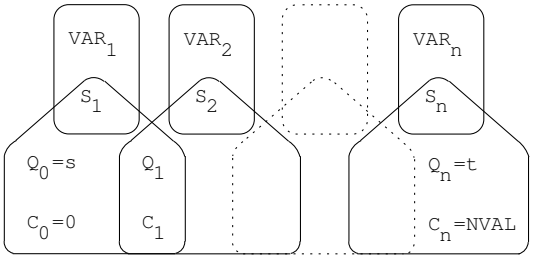


Figure 4.221: Hypergraph of the reformulation corresponding to the automaton of the NVAL parameter of the group_skip_isolated_item constraint

Key words timetabling constraint, strongly connected component, automaton, automaton with counters, alpha-acyclic constraint network(2), alpha-acyclic constraint network(3).

4.99 heighest_peak

Origin Derived from peak.

Constraint heighest_peak(HEIGHT, VARIABLES)

Argument(s)

HEIGHT	:	dvar
VARIABLES	:	collection(var − dvar)

Restriction(s)

HEIGHT ≥ 0
VARIABLES.var ≥ 0
required(VARIABLES, var)

Purpose

A variable V_k $(1 < k < m)$ of the sequence of variables VARIABLES $= V_1, \ldots, V_m$ is a *peak* if and only if there exist an i $(1 < i \leq k)$ such that $V_{i-1} < V_i$ and $V_i = V_{i+1} = \ldots = V_k$ 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

$$\texttt{heighest\_peak}\left(\begin{array}{l} 8, \left\{\begin{array}{l} \texttt{var}-1,\\ \texttt{var}-1,\\ \texttt{var}-4,\\ \texttt{var}-8,\\ \texttt{var}-6,\\ \texttt{var}-2,\\ \texttt{var}-7,\\ \texttt{var}-1 \end{array}\right\} \end{array}\right)$$

The previous constraint holds since 8 is the maximum peak of the sequence 1 1 4 8 6 2 7 1.

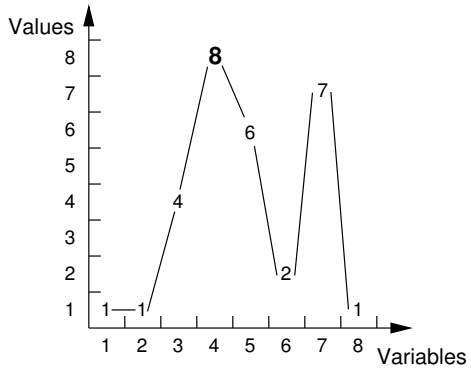


Figure 4.222: The sequence and its heighest peak

Automaton Figure 4.223 depicts the automaton associated to the heighest_peak constraint. To each pair of consecutive variables $(\texttt{VAR}_i, \texttt{VAR}_{i+1})$ of the collection VARIABLES corresponds a signature variable $\texttt{S}_i$. The following signature constraint links $\texttt{VAR}_i$, $\texttt{VAR}_{i+1}$ and $\texttt{S}_i$:
$\texttt{VAR}_i > \texttt{VAR}_{i+1} \Leftrightarrow \texttt{S}_i = 0 \wedge \texttt{VAR}_i = \texttt{VAR}_{i+1} \Leftrightarrow \texttt{S}_i = 1 \wedge \texttt{VAR}_i < \texttt{VAR}_{i+1} \Leftrightarrow \texttt{S}_i = 2$.

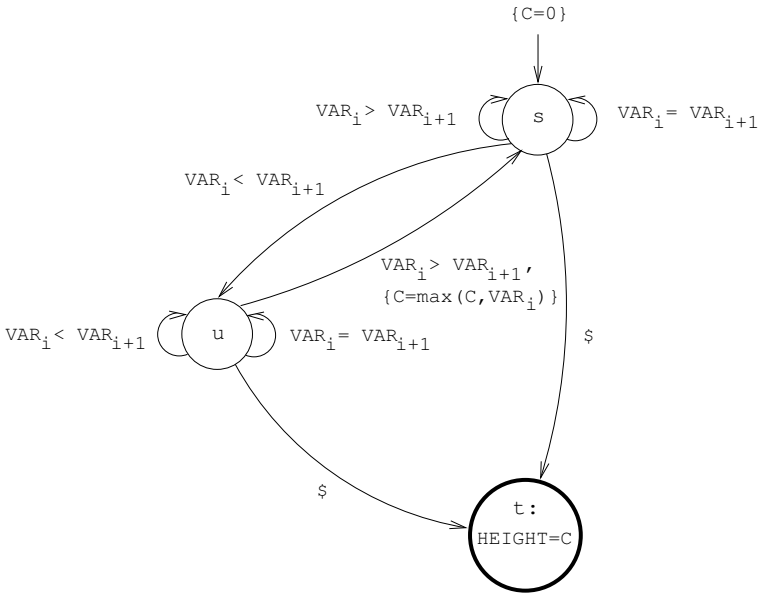


Figure 4.223: Automaton of the `heighest_peak` constraint

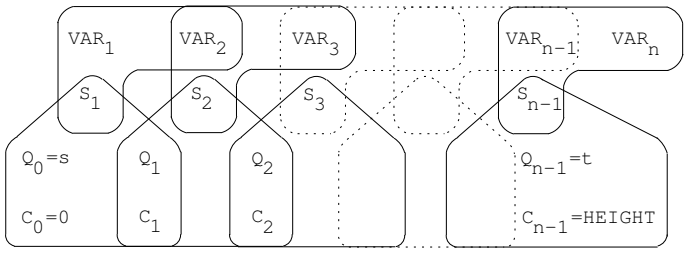


Figure 4.224: Hypergraph of the reformulation corresponding to the automaton of the `heighest_peak` constraint

See also peak, deepest_valley.

Key words sequence, automaton, automaton with counters, sliding cyclic(1) constraint network(2).

4.100 in

Origin Domain definition.

Constraint in(VAR, 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) col(VARIABLES − collection(var − dvar), [item(var − VAR)])

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) $\mathbf{NARC} = 1$

Example in(3, {val − 1, val − 3})

Parts (A) and (B) of Figure 4.225 respectively show the initial and final graph. Since we use the $\mathbf{NARC}$ graph property, the unique arc of the final graph is stressed in bold.

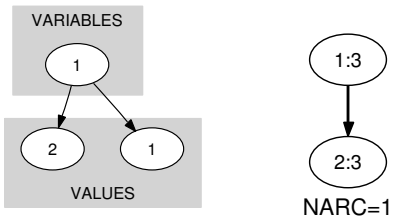


Figure 4.225: Initial and final graph of the in constraint

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 $\mathbf{NARC} = 1$ to $\mathbf{NARC} \geq 1$ and simplify $\overline{\mathbf{NARC}}$ to $\overline{\mathbf{NARC}}$.

Automaton Figure 4.226 depicts the automaton associated to the `in` constraint. Let $\texttt{VAL}_i$ be the `val` attribute of the i^{th} item of the `VALUES` collection. To each pair $(\texttt{VAR}, \texttt{VAL}_i)$ corresponds a 0-1 signature variable $\texttt{S}_i$ as well as the following signature constraint: $\texttt{VAR} = \texttt{VAL}_i \Leftrightarrow \texttt{S}_i$.

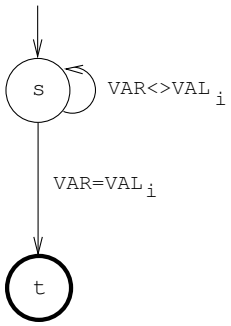


Figure 4.226: Automaton of the `in` constraint

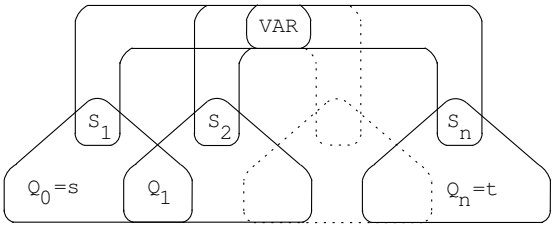


Figure 4.227: Hypergraph of the reformulation corresponding to the automaton of the `in` constraint

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 explicitely defined by tuples of values.

Constraint in_relation(VARIABLES, TUPLES_OF_VALS)

Synonym(s) extension.

Type(s)
TUPLE_OF_VARS : collection(var − dvar)
TUPLE_OF_VALS : collection(val − int)

Argument(s)
VARIABLES : TUPLE_OF_VARS
TUPLES_OF_VALS : collection(tuple − TUPLE_OF_VALS)

Restriction(s)
required(TUPLE_OF_VARS, var)
required(TUPLE_OF_VALS, val)
required(TUPLES_OF_VALS, tuple)
min_size(TUPLES_OF_VALS, tuple) = |VARIABLES|
max_size(TUPLES_OF_VALS, tuple) = |VARIABLES|

Purpose Enforce the tuple of variables VARIABLES to take its value out of a set of tuples of values TUPLES_OF_VALS. The *value* of a tuple of variables $\langle V_1, V_2, \ldots, V_n \rangle$ is a tuple of values $\langle U_1, U_2, \ldots, U_n \rangle$ if and only if $V_1 = U_1 \wedge V_2 = U_2 \wedge \cdots \wedge V_n = U_n$.

Derived Collection(s) col(TUPLES_OF_VARS − collection(vec − TUPLE_OF_VARS), [item(vec − VARIABLES)])

Arc input(s) TUPLES_OF_VARS TUPLES_OF_VALS

Arc generator $PRODUCT \mapsto$ collection(tuples_of_vars, tuples_of_vals)

Arc arity 2

Arc constraint(s) vec_eq_tuple(tuples_of_vars.vec, tuples_of_vals.tuple)

Graph property(ies) $\mathbf{NARC} \geq 1$

Example
$$\text{in\_relation}\left(\begin{array}{l}\{\text{var}-5, \text{var}-3, \text{var}-3\},\\ \left\{\begin{array}{l}\text{tuple}-\{\text{val}-5, \text{val}-2, \text{val}-3\},\\ \text{tuple}-\{\text{val}-5, \text{val}-2, \text{val}-6\},\\ \text{tuple}-\{\text{val}-5, \text{val}-3, \text{val}-3\}\end{array}\right\}\end{array}\right)$$

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.

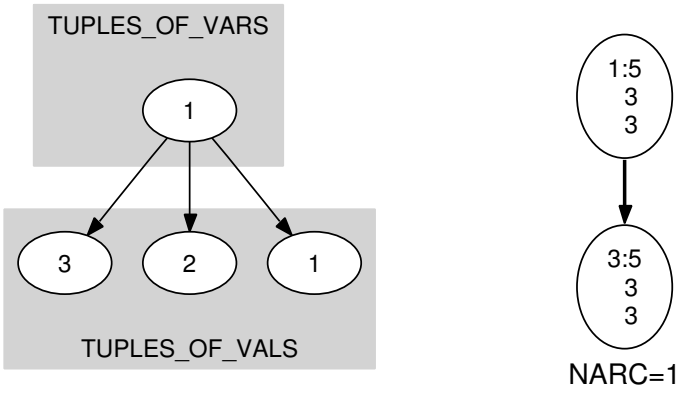


Figure 4.228: Initial and final graph of the `in_relation` constraint

4.102 in_same_partition

Origin Used for defining several entries of this catalog.

Constraint in_same_partition(VAR1, VAR2, PARTITIONS)

Type(s) VALUES : collection(val − int)

Argument(s)

VAR1	:	dvar
VAR2	:	dvar
PARTITIONS	:	collection(p − VALUES)

Restriction(s)
required(VALUES, val)
distinct(VALUES, val)
required(PARTITIONS, p)
|PARTITIONS| ≥ 2

Purpose 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.

Derived Collection(s) col(VARIABLES − collection(var − dvar), [item(var − VAR1), item(var − VAR2)])

Arc input(s) VARIABLES PARTITIONS

Arc generator $PRODUCT \mapsto$ collection(variables, partitions)

Arc arity 2

Arc constraint(s) in(variables.var, partitions.p)

Graph property(ies)
- **NSOURCE** $= 2$
- **NSINK** $= 1$

Example

$$\texttt{in\_same\_partition}\left(\begin{array}{l} 6, 2, \left\{\begin{array}{l} \texttt{p} - \{\texttt{val} - 1, \texttt{val} - 3\}, \\ \texttt{p} - \{\texttt{val} - 4\}, \\ \texttt{p} - \{\texttt{val} - 2, \texttt{val} - 6\} \end{array}\right\} \end{array}\right)$$

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.

Graph model 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.

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 $\mathbf{NSINK} = 1$ to $\mathbf{NSINK} \geq 1$ and simplify $\underline{\overline{\mathbf{NSINK}}}$ to $\overline{\mathbf{NSINK}}$.

Automaton Figure 4.230 depicts the automaton associated to the `in_same_partition` constraint. Let $\mathtt{VALUES}_i$ be the `p` attribute of the i^{th} item of the `PARTITIONS` collection. To each triple $(\mathtt{VAR1}, \mathtt{VAR2}, \mathtt{VALUES}_i)$ corresponds a 0-1 signature variable $\mathtt{S}_i$ as well as the following signature constraint: $((\mathtt{VAR1} \in \mathtt{VALUES}_i) \wedge (\mathtt{VAR2} \in \mathtt{VALUES}_i)) \Leftrightarrow \mathtt{S}_i$.

Used in `alldifferent_partition`, `balance_partition`, `change_partition`, `common_partition`, `nclass`, `same_partition`, `soft_same_partition_var`, `soft_used_by_partition_var`, `used_by_partition`.

See also `in`.

Key words value constraint, partition, automaton, automaton without counters, centered cyclic(2) constraint network(1), derived collection.

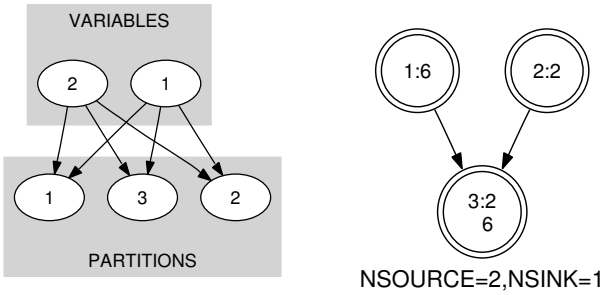


Figure 4.229: Initial and final graph of the `in_same_partition` constraint

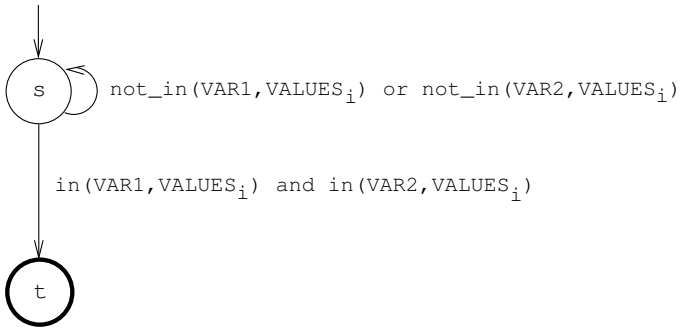


Figure 4.230: Automaton of the `in_same_partition` constraint

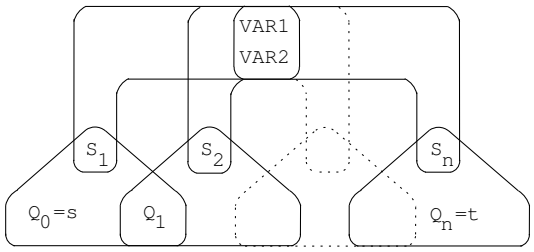


Figure 4.231: Hypergraph of the reformulation corresponding to the automaton of the `in_same_partition` constraint

4.103 in_set

Origin Used for defining constraints with set variables.

Constraint in_set(VAL, SET)

Argument(s)

VAL	:	dvar
SET	:	svar

Purpose Constraint variable VAL to belong to set SET.

Example in_set$(3, \{1, 3\})$

Used in clique, cutset, discrepancy, inverse_set, k_cut, link_set_to_booleans, path_from_to, strongly_connected, sum, sum_set, symmetric_cardinality, symmetric_gcc, tour.

Key words predefined constraint, value constraint, included, constraint involving set variables.

4.104 increasing

Origin KOALOG

Constraint increasing(VARIABLES)

Argument(s) VARIABLES : collection(var − dvar)

Restriction(s) |VARIABLES| > 0
required(VARIABLES, var)

Purpose The variables of the collection VARIABLES are increasing.

Arc input(s) VARIABLES

Arc generator $PATH \mapsto$ collection(variables1, variables2)

Arc arity 2

Arc constraint(s) variables1.var ≤ variables2.var

Graph property(ies) $\mathbf{NARC} = |\text{VARIABLES}| - 1$

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 $\mathbf{NARC}$ graph property, the arcs of the final graph are stressed in bold.

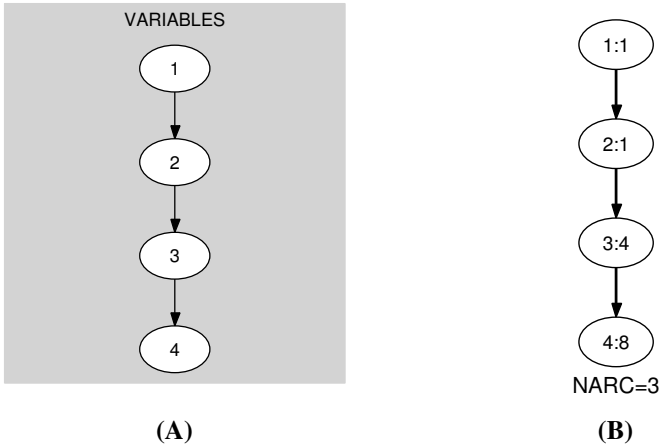


Figure 4.232: Initial and final graph of the increasing constraint

Automaton Figure 4.233 depicts the automaton associated to the `increasing` constraint. To each pair of consecutive variables $(\texttt{VAR}_i, \texttt{VAR}_{i+1})$ of the collection `VARIABLES` corresponds a 0-1 signature variable $\texttt{S}_i$. The following signature constraint links $\texttt{VAR}_i$, $\texttt{VAR}_{i+1}$ and $\texttt{S}_i$: $\texttt{VAR}_i > \texttt{VAR}_{i+1} \Leftrightarrow \texttt{S}_i$.

See also `strictly_increasing`, `decreasing`, `strictly_decreasing`.

Key words decomposition, order constraint, automaton, automaton without counters, sliding cyclic(1) constraint network(1).

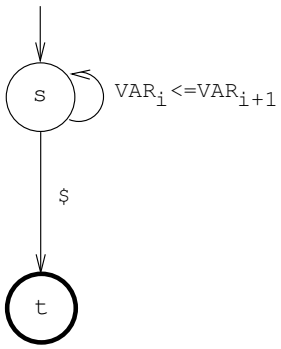


Figure 4.233: Automaton of the `increasing` constraint

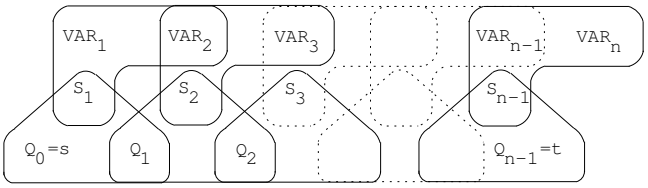


Figure 4.234: Hypergraph of the reformulation corresponding to the automaton of the `increasing` constraint

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

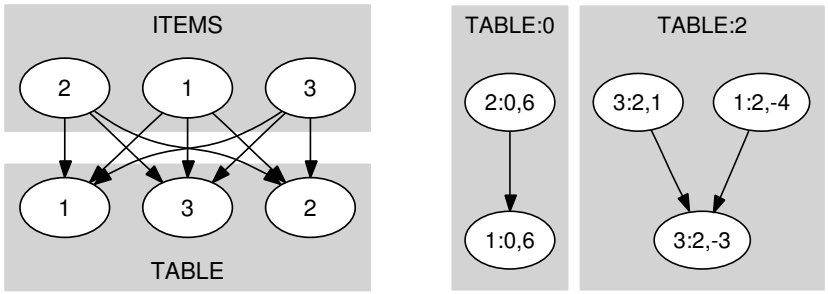
$$\text{SUCC} \mapsto \left[\begin{array}{l} \text{source}, \\ \text{variables} - \text{col}\left(\begin{array}{l} \text{VARIABLES} - \text{collection}(\text{var} - \text{dvar}), \\ [\text{item}(\text{var} - \text{ITEMS.weight})] \end{array} \right) \end{array} \right]$$

Constraint(s) on sets sum_ctr(variables, =, TABLE.sum)

Example

$$\text{indexed\_sum}\left(\begin{array}{l} \left\{ \begin{array}{ll} \text{index} - 2 & \text{weight} - -4, \\ \text{index} - 0 & \text{weight} - 6, \\ \text{index} - 2 & \text{weight} - 1 \end{array} \right\}, \\ \left\{ \begin{array}{ll} \text{index} - 0 & \text{sum} - 6, \\ \text{index} - 1 & \text{sum} - 0, \\ \text{index} - 2 & \text{sum} - -3 \end{array} \right\} \end{array} \right)$$

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.



(A) **(B)**

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 inflexion(N, VARIABLES)

Argument(s)

N	:	dvar
VARIABLES	:	collection(var − dvar)

Restriction(s)

$\texttt{N} \geq 1$
$\texttt{N} \leq |\texttt{VARIABLES}|$
required(VARIABLES, var)

Purpose

N is equal to the number of times that the following conjunctions of constraints hold:

- $X_i \texttt{CTR} X_{i+1} \wedge X_i \neq X_{i+1}$,
- $X_{i+1} = X_{i+2} \wedge \cdots \wedge X_{j-2} = X_{j-1}$,
- $X_{j-1} \neq X_j \wedge X_{j-1} \neg\texttt{CTR} X_j$.

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

$$\texttt{inflexion}\left(\begin{array}{l} 3, \left\{\begin{array}{l} \texttt{var}-1,\\ \texttt{var}-1,\\ \texttt{var}-4,\\ \texttt{var}-8,\\ \texttt{var}-8,\\ \texttt{var}-2,\\ \texttt{var}-7,\\ \texttt{var}-1 \end{array}\right\} \end{array}\right)$$

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.

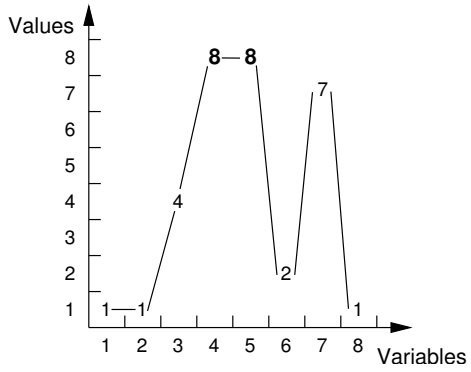


Figure 4.236: The sequence and its three inflexions

Automaton Figure 4.237 depicts the automaton associated to the `inflexion` constraint. To each pair of consecutive variables $(\mathtt{VAR}_i, \mathtt{VAR}_{i+1})$ of the collection `VARIABLES` corresponds a signature variable $\mathtt{S}_i$. The following signature constraint links $\mathtt{VAR}_i$, $\mathtt{VAR}_{i+1}$ and $\mathtt{S}_i$: $(\mathtt{VAR}_i > \mathtt{VAR}_{i+1} \Leftrightarrow \mathtt{S}_i = 0) \wedge (\mathtt{VAR}_i = \mathtt{VAR}_{i+1} \Leftrightarrow \mathtt{S}_i = 1) \wedge (\mathtt{VAR}_i < \mathtt{VAR}_{i+1} \Leftrightarrow \mathtt{S}_i = 2)$.

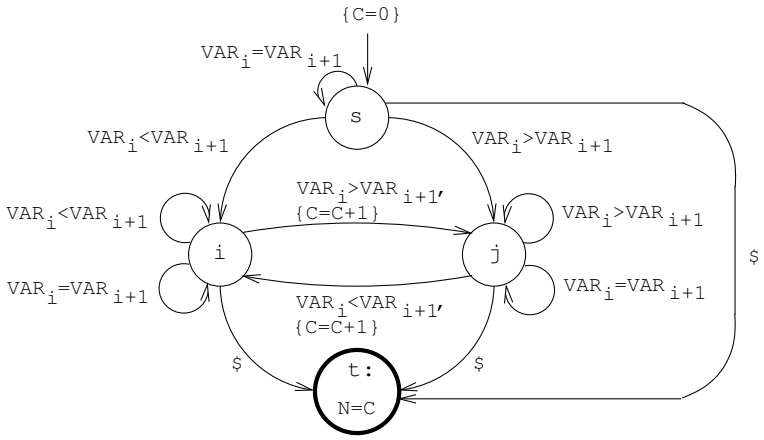


Figure 4.237: Automaton of the `inflexion` constraint

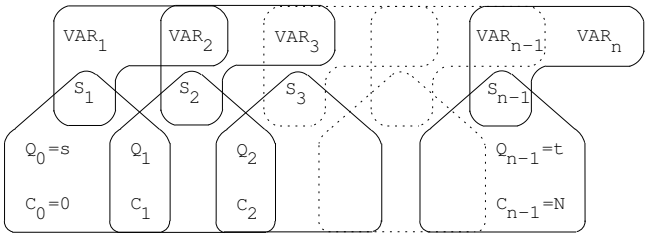


Figure 4.238: Hypergraph of the reformulation corresponding to the automaton of the `inflexion` constraint

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

$$\texttt{int\_value\_precede}\left(\begin{array}{c} 0, 1, \left\{\begin{array}{l}\texttt{var}-4,\\ \texttt{var}-0,\\ \texttt{var}-6,\\ \texttt{var}-1,\\ \texttt{var}-0\end{array}\right\}\end{array}\right)$$

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 $\texttt{VAR}_i$ be the i^{th} variable of the VARIABLES collection. To each triple $(\texttt{S}, \texttt{T}, \texttt{VAR}_i)$ corresponds a signature variable $\texttt{S}_i$ as well as the following signature constraint: $(\texttt{VAR}_i = \texttt{S} \Leftrightarrow \texttt{S}_i = 1) \wedge (\texttt{VAR}_i = \texttt{T} \Leftrightarrow \texttt{S}_i = 2) \wedge (\texttt{VAR}_i \neq \texttt{S} \wedge \texttt{VAR}_i \neq \texttt{T} \Leftrightarrow \texttt{S}_i = 3)$.

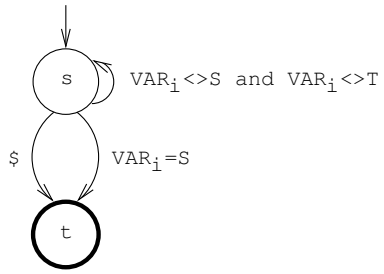


Figure 4.239: Automaton of the int_value_precede constraint

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.

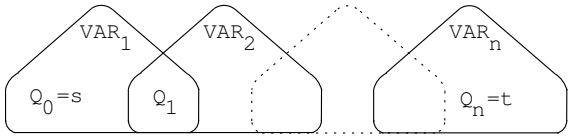


Figure 4.240: Hypergraph of the reformulation corresponding to the automaton of the `int_value_precede` constraint

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

$$\texttt{int\_value\_precede\_chain}\left(\begin{array}{l}\{\texttt{val}-4,\texttt{val}-0,\texttt{val}-1\},\\ \left\{\begin{array}{l}\texttt{var}-4,\\ \texttt{var}-0,\\ \texttt{var}-6,\\ \texttt{var}-1,\\ \texttt{var}-0\end{array}\right\}\end{array}\right)$$

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 $\texttt{VAR}_i$ be the i^{th} variable of the VARIABLES collection. Let $\texttt{VAL}_j$ $(1 < j < |\texttt{VALUES}|)$ denotes the j^{th} value of the VALUES collection. To each variable $\texttt{VAR}_i$ corresponds a signature variable $\texttt{S}_i$ as well as the following signature constraint: $(\texttt{VAR}_i \notin \texttt{VALUES} \Leftrightarrow \texttt{S}_i = 0) \wedge (\texttt{VAR}_i = \texttt{VAL}_1 \Leftrightarrow \texttt{S}_i = 1) \wedge (\texttt{VAR}_i = \texttt{VAL}_2 \Leftrightarrow \texttt{S}_i = 2) \wedge \cdots \wedge (\texttt{VAR}_i = \texttt{VAL}_{|\texttt{VALUES}|} \Leftrightarrow \texttt{S}_i = |\texttt{VALUES}|)$.

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.

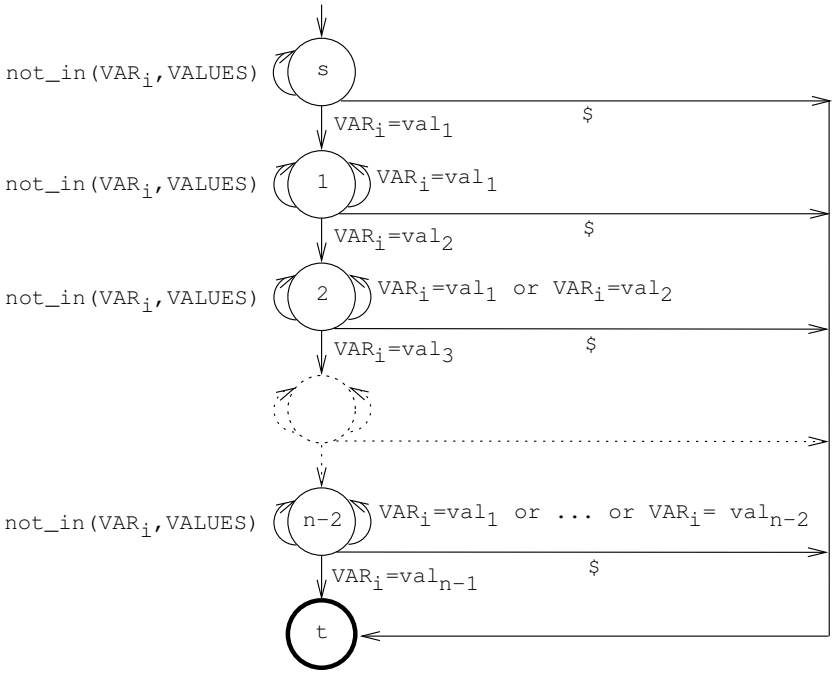


Figure 4.241: Automaton of the `int_value_precede_chain` constraint

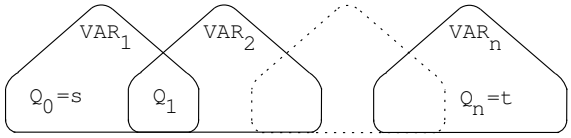


Figure 4.242: Hypergraph of the reformulation corresponding to the automaton of the `int_value_precede_chain` constraint

4.109 interval_and_count

Origin [122]

Constraint interval_and_count(ATMOST, COLOURS, TASKS, SIZE_INTERVAL)

Argument(s)

ATMOST	:	int
COLOURS	:	collection(val − int)
TASKS	:	collection(origin − dvar, colour − dvar)
SIZE_INTERVAL	:	int

Restriction(s)

ATMOST ≥ 0
required(COLOURS, val)
distinct(COLOURS, val)
required(TASKS, [origin, colour])
SIZE_INTERVAL > 0

Purpose

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 $[k \cdot$ SIZE_INTERVAL, $k \cdot$ SIZE_INTERVAL + 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.

Arc input(s) TASKS TASKS

Arc generator $PRODUCT \mapsto$ collection(tasks1, tasks2)

Arc arity 2

Arc constraint(s) tasks1.origin/SIZE_INTERVAL = tasks2.origin/SIZE_INTERVAL

Sets

$$\text{SUCC} \mapsto \left[\begin{array}{l} \texttt{source}, \\ \texttt{variables} - \texttt{col}\left(\begin{array}{l} \texttt{VARIABLES} - \texttt{collection}(\texttt{var} - \texttt{dvar}), \\ [\texttt{item}(\texttt{var} - \texttt{TASKS.colour})] \end{array}\right) \end{array}\right]$$

Constraint(s) on sets among_low_up(0, ATMOST, variables, COLOURS)

Example

$$\texttt{interval\_and\_count}\left(\begin{array}{l} 2, \{\texttt{val} - 4\}, \\ \left\{\begin{array}{ll} \texttt{origin} - 1 & \texttt{colour} - 4, \\ \texttt{origin} - 0 & \texttt{colour} - 9, \\ \texttt{origin} - 10 & \texttt{colour} - 4, \\ \texttt{origin} - 4 & \texttt{colour} - 4 \end{array}\right\}, 5 \end{array}\right)$$

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.

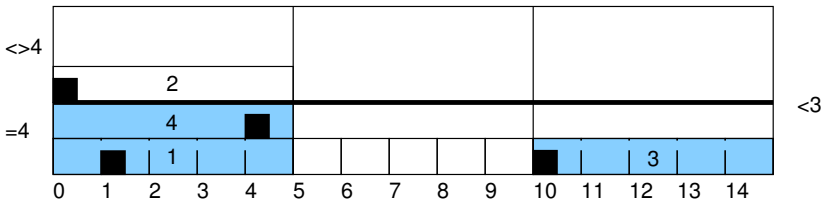
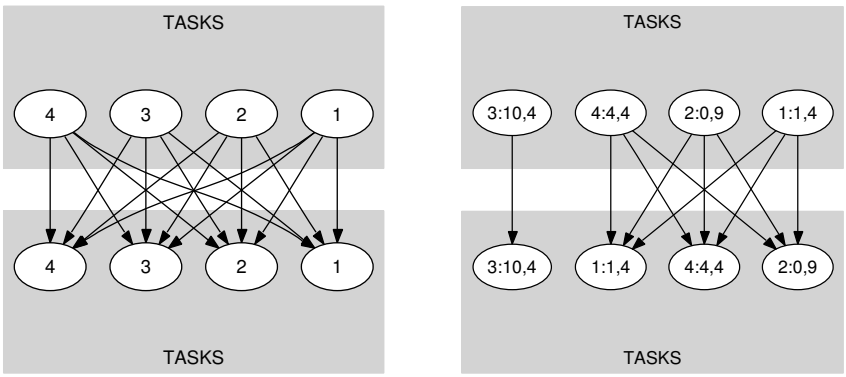


Figure 4.243: Solution with the use of each interval



(A) **(B)**

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 $\mathcal{S}$ of successors of the different vertices of the final graph. This put a restriction on the maximum number of tasks of $\mathcal{S}$ 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 $\texttt{COLOUR}_i$ be the colour attribute of the i^{th} item of the TASKS collection. To each pair $(\texttt{COLOURS}, \texttt{COLOUR}_i)$ corresponds a signature variable $\texttt{S}_i$ as well as the following signature constraint: $\texttt{COLOUR}_i \in \texttt{COLOURS} \Leftrightarrow \texttt{S}_i$.

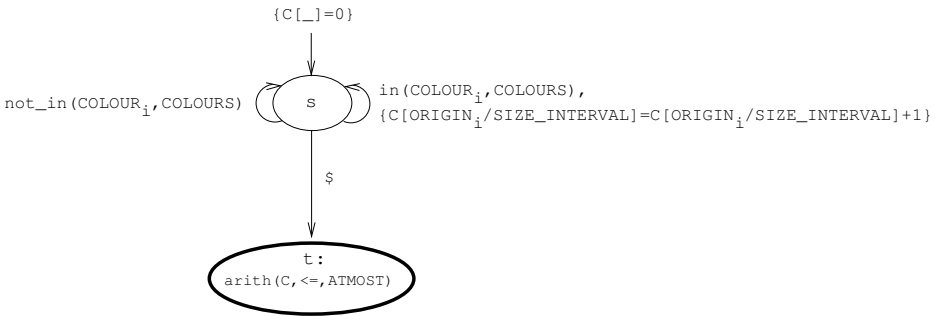


Figure 4.245: Automaton of the interval_and_count constraint

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 consecutives 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: $(\texttt{tasks1.origin}/\texttt{SIZE\_INTERVAL}) \equiv \texttt{REST}(\bmod \texttt{QUOTIENT})$

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.

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: $[k \cdot$ SIZE_INTERVAL$, k \cdot$ SIZE_INTERVAL $+$ SIZE_INTERVAL $- 1]$, where k is an integer.

Arc input(s) TASKS TASKS

Arc generator $PRODUCT \mapsto$ collection(tasks1, tasks2)

Arc arity 2

Arc constraint(s) tasks1.origin/SIZE_INTERVAL = tasks2.origin/SIZE_INTERVAL

Sets

$$\text{SUCC} \mapsto \left[\begin{array}{l} \texttt{source}, \\ \texttt{variables} - \texttt{col}\left(\begin{array}{l} \texttt{VARIABLES} - \texttt{collection}(\texttt{var} - \texttt{dvar}), \\ [\texttt{item}(\texttt{var} - \texttt{TASKS.height})] \end{array} \right) \end{array} \right]$$

Constraint(s) on sets sum_ctr(variables, $\leq$, LIMIT)

Example

$$\texttt{interval\_and\_sum}\left(5, \left\{ \begin{array}{ll} \texttt{origin} - 1 & \texttt{height} - 2, \\ \texttt{origin} - 10 & \texttt{height} - 2, \\ \texttt{origin} - 10 & \texttt{height} - 3, \\ \texttt{origin} - 4 & \texttt{height} - 1 \end{array} \right\}, 5 \right)$$

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.

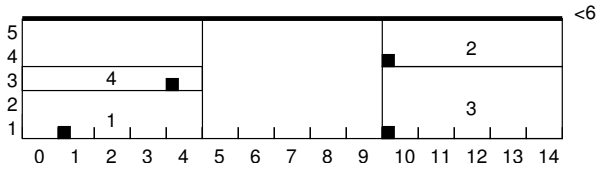
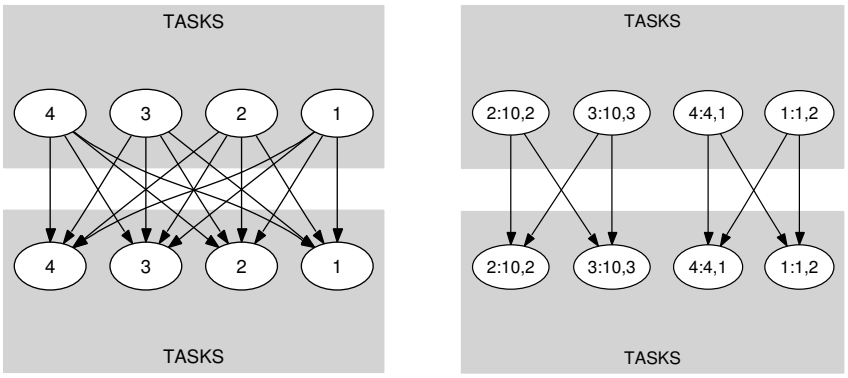


Figure 4.246: Solution showing for each interval the corresponding tasks



(A) **(B)**

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.

4.111 inverse

Origin CHIP

Constraint inverse(NODES)

Synonym(s) assignment.

Argument(s) NODES : collection(index − int, succ − dvar, pred − dvar)

Restriction(s)
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|

Purpose 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.

Arc input(s) NODES

Arc generator $CLIQUE \mapsto$ collection(nodes1, nodes2)

Arc arity 2

Arc constraint(s)
- nodes1.succ = nodes2.index
- nodes2.pred = nodes1.index

Graph property(ies) **NARC** = |NODES|

Example
$$\texttt{inverse}\left(\left\{\begin{array}{lll}\texttt{index}-1 & \texttt{succ}-2 & \texttt{pred}-2,\\ \texttt{index}-2 & \texttt{succ}-1 & \texttt{pred}-1,\\ \texttt{index}-3 & \texttt{succ}-5 & \texttt{pred}-4,\\ \texttt{index}-4 & \texttt{succ}-3 & \texttt{pred}-5,\\ \texttt{index}-5 & \texttt{succ}-4 & \texttt{pred}-3\end{array}\right\}\right)$$

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.

Graph model 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:

- 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 $\mathbf{NARC} = |\texttt{NODES}|$ to $\mathbf{NARC} \geq |\texttt{NODES}|$ and simplify $\underline{\overline{\mathbf{NARC}}}$ to $\overline{\mathbf{NARC}}$.

Automaton

Figure 4.250 depicts the automaton associated to the `inverse` constraint. To each item of the collection `NODES` corresponds a signature variable $\texttt{S}_i$, 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* an *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.

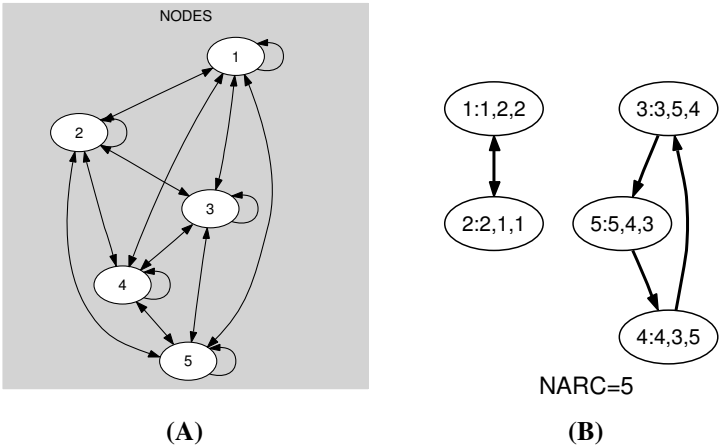


Figure 4.249: Initial and final graph of the `inverse` constraint

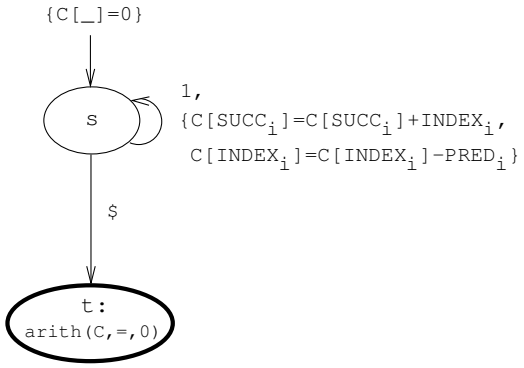


Figure 4.250: Automaton of the `inverse` constraint

4.112 inverse_set

Origin Derived from inverse.

Constraint inverse_set(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 $PRODUCT \mapsto$ collection(x, y)

Arc arity 2

Arc constraint(s) in_set(y.index, x.set) ⇔ in_set(x.index, y.set)

Graph property(ies) $\mathbf{NARC} = |\mathtt{X}| * |\mathtt{Y}|$

Example

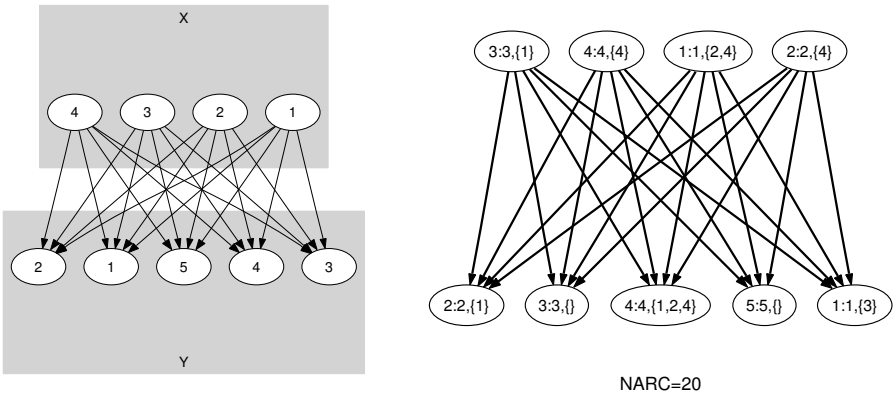
$$\texttt{inverse\_set}\left(\left\{\begin{array}{ll}\texttt{index}-1 & \texttt{set}-\{2,4\},\\ \texttt{index}-2 & \texttt{set}-\{4\},\\ \texttt{index}-3 & \texttt{set}-\{1\},\\ \texttt{index}-4 & \texttt{set}-\{4\}\end{array}\right\},\left\{\begin{array}{ll}\texttt{index}-1 & \texttt{set}-\{3\},\\ \texttt{index}-2 & \texttt{set}-\{1\},\\ \texttt{index}-3 & \texttt{set}-\emptyset,\\ \texttt{index}-4 & \texttt{set}-\{1,2,4\},\\ \texttt{index}-5 & \texttt{set}-\emptyset\end{array}\right\}\right)$$

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.



(A) **(B)**

Figure 4.251: Initial and final graph of the `inverse_set` constraint

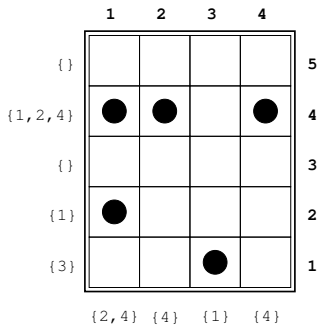


Figure 4.252: Board associated to the example

4.113 ith_pos_different_from_0

Origin Used for defining the automaton of min_n.

Constraint ith_pos_different_from_0(ITH, POS, VARIABLES)

Argument(s)

ITH	:	int
POS	:	dvar
VARIABLES	:	collection(var − dvar)

Restriction(s)

ITH ≥ 1
ITH $\leq |$VARIABLES$|$
POS $\geq$ ITH
POS $\leq |$VARIABLES$|$
required(VARIABLES, var)

Purpose POS is the position of the ITHth non-zero item of the sequence of variables VARIABLES.

Example

$$\texttt{ith\_pos\_different\_from\_0}\left(\begin{array}{l} 2, 4, \left\{\begin{array}{l}\texttt{var}-3,\\ \texttt{var}-0,\\ \texttt{var}-0,\\ \texttt{var}-8,\\ \texttt{var}-6\end{array}\right\}\end{array}\right)$$

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$_i$.

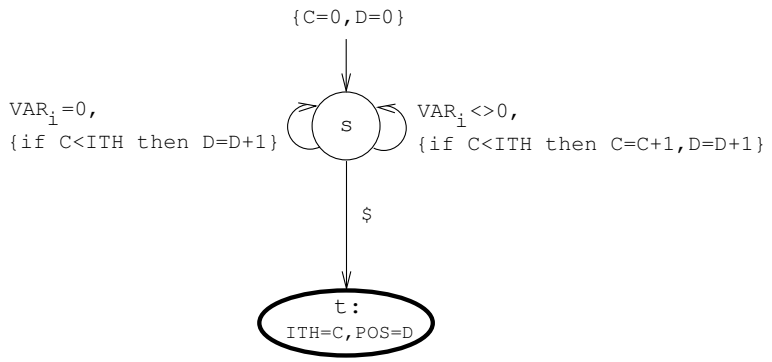


Figure 4.253: Automaton of the ith_pos_different_from_0 constraint

See also min_n.

Key words data constraint, table, joker value, automaton, automaton with counters, alpha-acyclic constraint network(3).

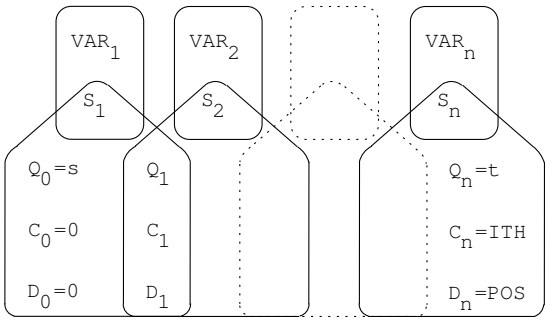


Figure 4.254: Hypergraph of the reformulation corresponding to the automaton of the `ith_pos_different_from_0` constraint

4.114 k_cut

Origin E. Althaus

Constraint k_cut(K, NODES)

Argument(s)

K	:	int
NODES	:	collection(index − int, succ − svar)

Restriction(s)

K ≥ 1
K ≤ |NODES|
required(NODES, [index, succ])
NODES.index ≥ 1
NODES.index ≤ |NODES|
distinct(NODES, index)

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) NODES

Arc generator $CLIQUE \mapsto$ collection(nodes1, nodes2)

Arc arity 2

Arc constraint(s) nodes1.index = nodes2.index ∨ in_set(nodes2.index, nodes1.succ)

Graph property(ies) **NCC** ≥ K

Example $\texttt{k\_cut}\left(3, \left\{\begin{array}{ll} \texttt{index} - 1 & \texttt{succ} - \emptyset, \\ \texttt{index} - 2 & \texttt{succ} - \{3,5\}, \\ \texttt{index} - 3 & \texttt{succ} - \{5\}, \\ \texttt{index} - 4 & \texttt{succ} - \emptyset, \\ \texttt{index} - 5 & \texttt{succ} - \{2,3\} \end{array}\right\}\right)$

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 succ attribute of a given vertex. Part (B) of Figure 4.255 gives the final graph associated to the example. The k_cut constraint holds since we have at least K = 3 connected components in the final graph.

Graph model nodes1.index = nodes2.index holds if nodes1 and nodes2 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 link_set_to_booleans.

Key words graph constraint, linear programming, connected component, constraint involving set variables.

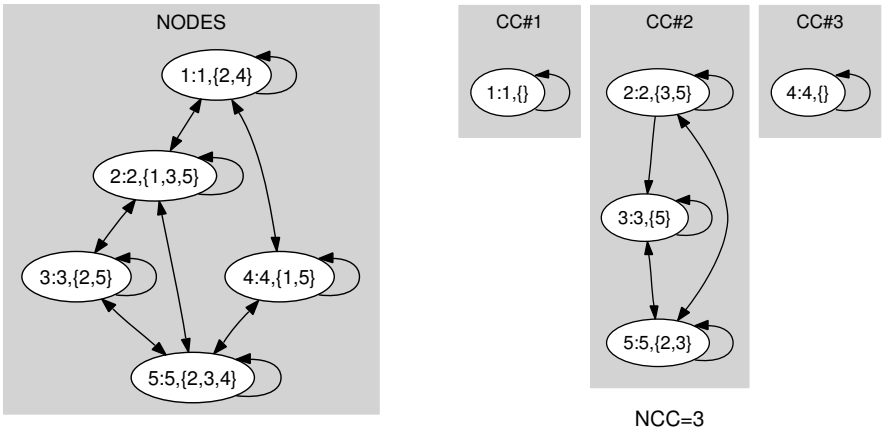


Figure 4.255: Initial and final graph of the `k_cut` set constraint

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 $\texttt{lex2}\left(\left\{\begin{array}{l}\texttt{vec} - \{\texttt{var} - 2, \texttt{var} - 2, \texttt{var} - 3\},\\ \texttt{vec} - \{\texttt{var} - 2, \texttt{var} - 3, \texttt{var} - 1\}\end{array}\right\}\right)$

Usage A *symmetry-breaking* constraint.

Remark The idea of this *symmetry-breaking* constraint can allready 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 lex_alldifferent(VECTORS)

Synonym(s) lex_alldiff, lex_alldistinct.

Type(s) VECTOR : collection(var − dvar)

Argument(s) VECTORS : collection(vec − VECTOR)

Restriction(s) required(VECTOR, var)
required(VECTORS, vec)
same_size(VECTORS, vec)

Purpose All the vectors of the collection VECTORS are distinct. Two vectors $(u_1, u_2, \ldots, u_n)$ and $(v_1, v_2, \ldots, v_n)$ are distinct if and only if there exist $i \in [1, n]$ such that $u_i \neq v_i$.

Arc input(s) VECTORS

Arc generator $CLIQUE(<) \mapsto$ collection(vectors1, vectors2)

Arc arity 2

Arc constraint(s) lex_different(vectors1.vec, vectors2.vec)

Graph property(ies) $\mathbf{NARC} = |\texttt{VECTORS}| * (|\texttt{VECTORS}| - 1)/2$

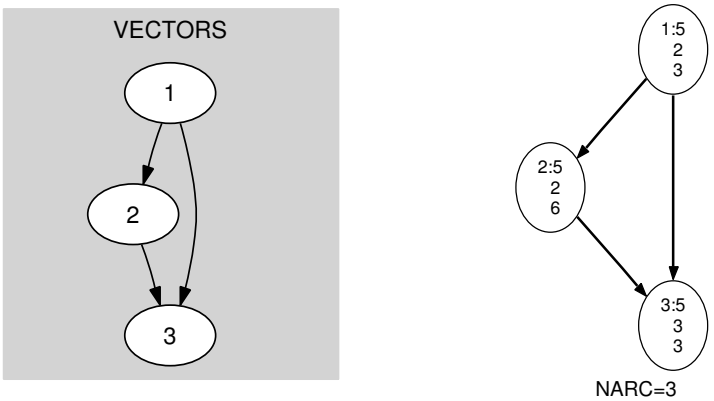
Example $\texttt{lex\_alldifferent}\left(\left\{\begin{array}{l}\texttt{vec} - \{\texttt{var} - 5, \texttt{var} - 2, \texttt{var} - 3\},\\ \texttt{vec} - \{\texttt{var} - 5, \texttt{var} - 2, \texttt{var} - 6\},\\ \texttt{vec} - \{\texttt{var} - 5, \texttt{var} - 3, \texttt{var} - 3\}\end{array}\right\}\right)$

Parts (A) and (B) of Figure 4.256 respectively show the initial and final graph. Since we use the $\mathbf{NARC}$ graph property, the arcs of the final graph are stressed in bold.

Signature Since we use the $CLIQUE(<)$ arc generator on the VECTORS collection the number of arcs of the initial graph is equal to $|\texttt{VECTORS}| \cdot (|\texttt{VECTORS}| - 1)/2$. For this reason we can rewrite $\mathbf{NARC} = |\texttt{VECTORS}| \cdot (|\texttt{VECTORS}| - 1)/2$ to $\mathbf{NARC} \geq |\texttt{VECTORS}| \cdot (|\texttt{VECTORS}| - 1)/2$ and simplify $\underline{\overline{\mathbf{NARC}}}$ to $\overline{\mathbf{NARC}}$.

See also alldifferent, lex_different.

Key words decomposition, vector, bipartite matching.



(A) **(B)**

Figure 4.256: Initial and final graph of the `lex_alldifferent` constraint

4.117 lex_between

Origin [126]

Constraint lex_between(LOWER_BOUND, VECTOR, UPPER_BOUND)

Argument(s)

LOWER_BOUND	:	collection(var − int)
VECTOR	:	collection(var − dvar)
UPPER_BOUND	:	collection(var − int)

Restriction(s)

required(LOWER_BOUND, var)
required(VECTOR, var)
required(UPPER_BOUND, var)
|LOWER_BOUND| = |VECTOR|
|UPPER_BOUND| = |VECTOR|
lex_lesseq(LOWER_BOUND, VECTOR)
lex_lesseq(VECTOR, UPPER_BOUND)

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

$$\texttt{lex\_between}\left(\begin{array}{l}\{\texttt{var}-5,\texttt{var}-2,\texttt{var}-3,\texttt{var}-9\},\\ \{\texttt{var}-5,\texttt{var}-2,\texttt{var}-6,\texttt{var}-2\},\\ \{\texttt{var}-5,\texttt{var}-2,\texttt{var}-6,\texttt{var}-3\}\end{array}\right)$$

Automaton Figure 4.257 depicts the automaton associated to the lex_between constraint. Let $\texttt{L}_i$, $\texttt{V}_i$ and $\texttt{U}_i$ respectively be the var attributes of the i^{th} items of the LOWER_BOUND, the VECTOR and the UPPER_BOUND collections. To each triple $(\texttt{L}_i, \texttt{V}_i, \texttt{U}_i)$ corresponds a signature variable $\texttt{S}_i$ as well as the following signature constraint:

$$(\texttt{L}_i < \texttt{V}_i) \wedge (\texttt{V}_i < \texttt{U}_i) \Leftrightarrow \texttt{S}_i = 0 \wedge$$
$$(\texttt{L}_i < \texttt{V}_i) \wedge (\texttt{V}_i = \texttt{U}_i) \Leftrightarrow \texttt{S}_i = 1 \wedge$$
$$(\texttt{L}_i < \texttt{V}_i) \wedge (\texttt{V}_i > \texttt{U}_i) \Leftrightarrow \texttt{S}_i = 2 \wedge$$
$$(\texttt{L}_i = \texttt{V}_i) \wedge (\texttt{V}_i < \texttt{U}_i) \Leftrightarrow \texttt{S}_i = 3 \wedge$$
$$(\texttt{L}_i = \texttt{V}_i) \wedge (\texttt{V}_i = \texttt{U}_i) \Leftrightarrow \texttt{S}_i = 4 \wedge$$
$$(\texttt{L}_i = \texttt{V}_i) \wedge (\texttt{V}_i > \texttt{U}_i) \Leftrightarrow \texttt{S}_i = 5 \wedge$$
$$(\texttt{L}_i > \texttt{V}_i) \wedge (\texttt{V}_i < \texttt{U}_i) \Leftrightarrow \texttt{S}_i = 6 \wedge$$
$$(\texttt{L}_i > \texttt{V}_i) \wedge (\texttt{V}_i = \texttt{U}_i) \Leftrightarrow \texttt{S}_i = 7 \wedge$$
$$(\texttt{L}_i > \texttt{V}_i) \wedge (\texttt{V}_i > \texttt{U}_i) \Leftrightarrow \texttt{S}_i = 8.$$

Usage This constraint does usually not occur explicitly in practice. However it shows up indirectly in the context of the lex_chain_less and the lex_chain_lesseq constraints: In order to have a complete filtering algorithm for the lex_chain_less and the lex_chain_lesseq constraints one has to come up with a complete filtering algorithm for the lex_between

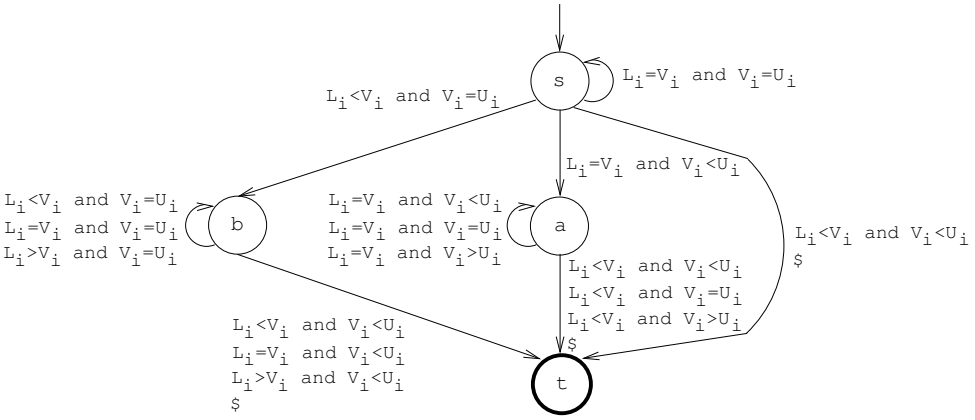


Figure 4.257: Automaton of the `lex_between` constraint

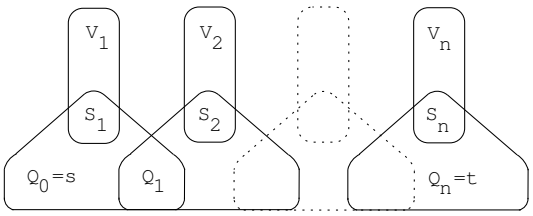


Figure 4.258: Hypergraph of the reformulation corresponding to the automaton of the `lex_between` constraint

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.

4.118 lex_chain_less

Origin [126]

Constraint lex_chain_less(VECTORS)

Usual name lex_chain

Type(s) VECTOR : collection(var − dvar)

Argument(s) VECTORS : collection(vec − VECTOR)

Restriction(s) required(VECTOR, var)
required(VECTORS, vec)
same_size(VECTORS, vec)

Purpose For each pair of consecutive vectors $\texttt{VECTOR}_i$ and $\texttt{VECTOR}_{i+1}$ of the VECTORS collection we have that $\texttt{VECTOR}_i$ is lexicographically strictly less than $\texttt{VECTOR}_{i+1}$. Given two vectors, $\vec{X}$ and $\vec{Y}$ of n components, $\langle X_0, \ldots, X_n \rangle$ and $\langle Y_0, \ldots, Y_n \rangle$, $\vec{X}$ is *lexicographically strictly less than* $\vec{Y}$ if and only if $X_0 < Y_0$ or $X_0 = Y_0$ and $\langle X_1, \ldots, X_n \rangle$ is lexicographically strictly less than $\langle Y_1, \ldots, Y_n \rangle$.

Arc input(s) VECTORS

Arc generator $PATH \mapsto$ collection(vectors1, vectors2)

Arc arity 2

Arc constraint(s) lex_less(vectors1.vec, vectors2.vec)

Graph property(ies) $\mathbf{NARC} = |\texttt{VECTORS}| - 1$

Example

$$\texttt{lex\_chain\_less}\left(\left\{\left\{\texttt{vec} - \left\{\begin{array}{l}\texttt{var}-5,\\ \texttt{var}-2,\\ \texttt{var}-3,\\ \texttt{var}-9\end{array}\right\},\right\},\left\{\texttt{vec} - \left\{\begin{array}{l}\texttt{var}-5,\\ \texttt{var}-2,\\ \texttt{var}-6,\\ \texttt{var}-2\end{array}\right\},\right\},\left\{\texttt{vec} - \left\{\begin{array}{l}\texttt{var}-5,\\ \texttt{var}-2,\\ \texttt{var}-6,\\ \texttt{var}-3\end{array}\right\}\right\}\right\}\right)$$

Parts (A) and (B) of Figure 4.259 respectively show the initial and final graph. Since we use the $\mathbf{NARC}$ 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 $PATH$ arc generator on the VECTORS collection the number of arcs of the initial graph is equal to |VECTORS| − 1. For this reason we can rewrite $\mathbf{NARC} = |\texttt{VECTORS}| - 1$ to $\mathbf{NARC} \geq |\texttt{VECTORS}| - 1$ and simplify $\underline{\mathbf{NARC}}$ to $\overline{\mathbf{NARC}}$.

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 allows to come up with a complete pruning.

Algorithm A complete filtering algorithm for a chain of lexicographical constraints is presented in [126].

See also lex_between, lex_chain_lesseq, lex_less, lex_lesseq, lex_greater, lex_greatereq.

Key words decomposition, order constraint, vector, symmetry, matrix symmetry, lexicographic order.

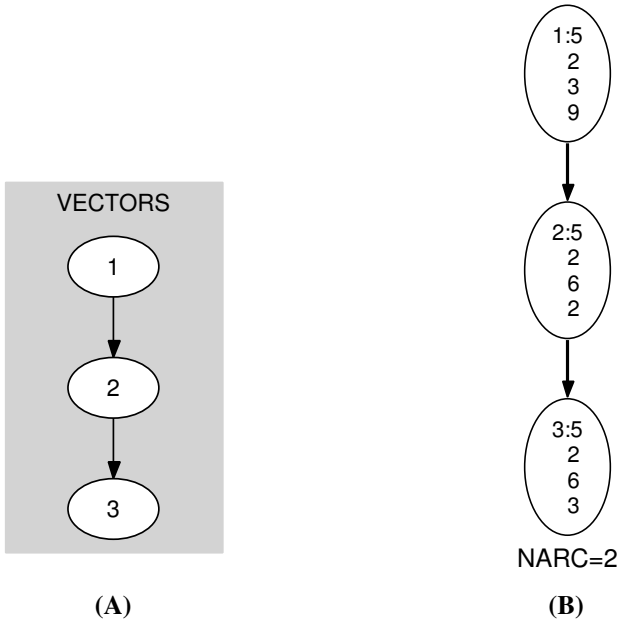


Figure 4.259: Initial and final graph of the `lex_chain_less` constraint

4.119 lex_chain_lesseq

Origin [126]

Constraint lex_chain_lesseq(VECTORS)

Usual name lex_chain

Type(s) VECTOR : collection(var − dvar)

Argument(s) VECTORS : collection(vec − VECTOR)

Restriction(s) required(VECTOR, var)
required(VECTORS, vec)
same_size(VECTORS, vec)

Purpose For each pair of consecutive vectors $\texttt{VECTOR}_i$ and $\texttt{VECTOR}_{i+1}$ of the VECTORS collection we have that $\texttt{VECTOR}_i$ is lexicographically less than or equal to $\texttt{VECTOR}_{i+1}$. Given two vectors, $\vec{X}$ and $\vec{Y}$ of n components, $\langle X_0, \ldots, X_n \rangle$ and $\langle Y_0, \ldots, Y_n \rangle$, $\vec{X}$ 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 $\langle X_1, \ldots, X_n \rangle$ is lexicographically less than or equal to $\langle Y_1, \ldots, Y_n \rangle$.

Arc input(s) VECTORS

Arc generator $PATH \mapsto$ collection(vectors1, vectors2)

Arc arity 2

Arc constraint(s) lex_lesseq(vectors1.vec, vectors2.vec)

Graph property(ies) $\mathbf{NARC} = |\texttt{VECTORS}| - 1$

Example

$$\texttt{lex\_chain\_lesseq}\left(\left\{\begin{array}{l}\{\texttt{vec} - \{\texttt{var} - 5, \texttt{var} - 2, \texttt{var} - 3, \texttt{var} - 9\}\},\\ \{\texttt{vec} - \{\texttt{var} - 5, \texttt{var} - 2, \texttt{var} - 6, \texttt{var} - 2\}\},\\ \{\texttt{vec} - \{\texttt{var} - 5, \texttt{var} - 2, \texttt{var} - 6, \texttt{var} - 2\}\}\end{array}\right\}\right)$$

Parts (A) and (B) of Figure 4.260 respectively show the initial and final graph. Since we use the $\mathbf{NARC}$ 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 $PATH$ arc generator on the VECTORS collection the number of arcs of the initial graph is equal to |VECTORS| − 1. For this reason we can rewrite $\mathbf{NARC} =$ |VECTORS| − 1 to $\mathbf{NARC} \geq$ |VECTORS| − 1 and simplify $\underline{\mathbf{NARC}}$ to $\overline{\mathbf{NARC}}$.

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 allows to come up with a complete pruning.

Algorithm A complete filtering algorithm for a chain of lexicographical constraints is presented in [126].

See also lex_between, lex_chain_less, lex_less, lex_lesseq, lex_greater, lex_greatereq.

Key words decomposition, order constraint, vector, symmetry, matrix symmetry, lexicographic order.

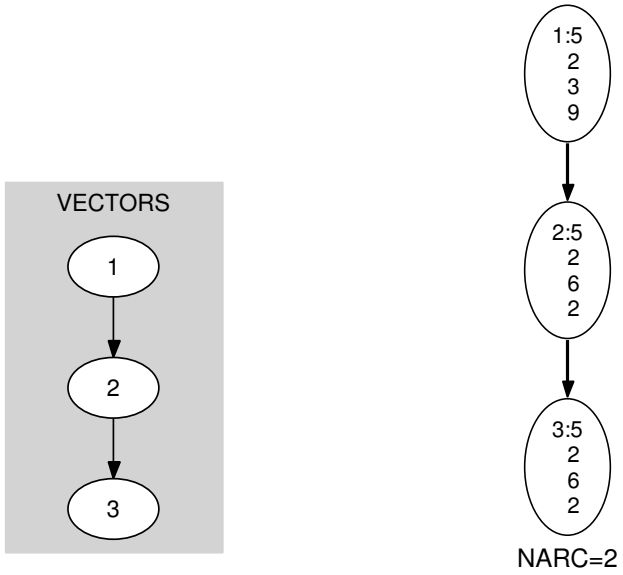


Figure 4.260: Initial and final graph of the `lex_chain_lesseq` constraint

4.120 lex_different

Origin Used for defining lex_alldifferent.

Constraint lex_different(VECTOR1, VECTOR2)

Argument(s)

VECTOR1	:	collection(var − dvar)
VECTOR2	:	collection(var − dvar)

Restriction(s) required(VECTOR1, var)
required(VECTOR2, var)
|VECTOR1| = |VECTOR2|

Purpose Vectors VECTOR1 and VECTOR2 differ from at least one component.

Arc input(s) VECTOR1 VECTOR2

Arc generator $PRODUCT(=) \mapsto$ collection(vector1, vector2)

Arc arity 2

Arc constraint(s) vector1.var ≠ vector2.var

Graph property(ies) $\mathbf{NARC} \geq 1$

Example $\texttt{lex\_different}\left(\begin{array}{l}\{\texttt{var}-5,\texttt{var}-2,\texttt{var}-7,\texttt{var}-1\},\\ \{\texttt{var}-5,\texttt{var}-3,\texttt{var}-7,\texttt{var}-1\}\end{array}\right)$

Parts (A) and (B) of Figure 4.261 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. It corresponds to a component where the two vectors differ.

Automaton Figure 4.262 depicts the automaton associated to the lex_different constraint. Let $\texttt{VAR1}_i$ and $\texttt{VAR2}_i$ respectively be the var attributes of the i^{th} items of the VECTOR1 and the VECTOR2 collections. To each pair $(\texttt{VAR1}_i, \texttt{VAR2}_i)$ corresponds a 0-1 signature variable $\texttt{S}_i$ as well as the following signature constraint: $\texttt{VAR1}_i = \texttt{VAR2}_i \Leftrightarrow \texttt{S}_i$.

Used in lex_alldifferent.

See also lex_greatereq, lex_less, lex_lesseq.

Key words vector, disequality, Berge-acyclic constraint network, automaton, automaton without counters.

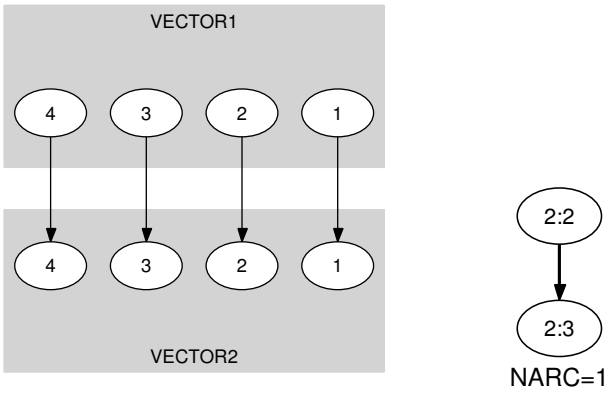


Figure 4.261: Initial and final graph of the `lex_different` constraint

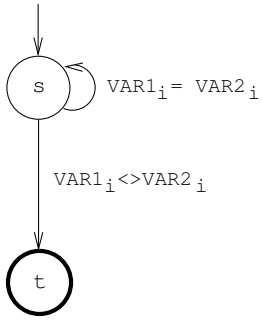


Figure 4.262: Automaton of the `lex_different` constraint

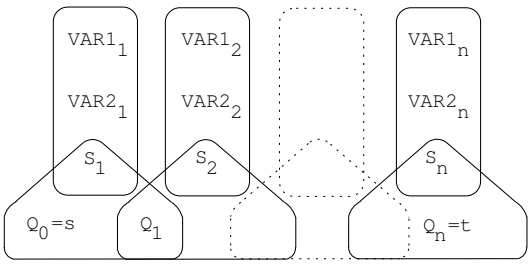


Figure 4.263: Hypergraph of the reformulation corresponding to the automaton of the `lex_different` constraint

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, $\langle X_0, \ldots, X_n \rangle$ and $\langle Y_0, \ldots, Y_n \rangle$, $\vec{X}$ is *lexicographically strictly greater than* $\vec{Y}$ if and only if $X_0 > Y_0$ or $X_0 = Y_0$ and $\langle X_1, \ldots, X_n \rangle$ is lexicographically strictly greater than $\langle Y_1, \ldots, Y_n \rangle$.

Derived Collection(s)

$$\texttt{col}\left(\begin{array}{l}\texttt{DESTINATION} - \texttt{collection}(\texttt{index} - \texttt{int}, \texttt{x} - \texttt{int}, \texttt{y} - \texttt{int}),\\ [\texttt{item}(\texttt{index} - 0, \texttt{x} - 0, \texttt{y} - 0)]\end{array}\right)$$

$$\texttt{col}\left(\begin{array}{l}\texttt{COMPONENTS} - \texttt{collection}(\texttt{index} - \texttt{int}, \texttt{x} - \texttt{dvar}, \texttt{y} - \texttt{dvar}),\\ [\texttt{item}(\texttt{index} - \texttt{VECTOR1.key}, \texttt{x} - \texttt{VECTOR1.var}, \texttt{y} - \texttt{VECTOR2.var})]\end{array}\right)$$

Arc input(s) COMPONENTS DESTINATION

Arc generator $PRODUCT(PATH, VOID) \mapsto \texttt{collection}(\texttt{item1}, \texttt{item2})$

Arc arity 2

Arc constraint(s) $\texttt{item2.index} > 0 \wedge \texttt{item1.x} = \texttt{item1.y} \vee \texttt{item2.index} = 0 \wedge \texttt{item1.x} > \texttt{item1.y}$

Graph property(ies) $\mathbf{PATH\_FROM\_TO}(\texttt{index}, 1, 0) = 1$

Example

$$\texttt{lex\_greater}\left(\begin{array}{l}\{\texttt{var} - 5, \texttt{var} - 2, \texttt{var} - 7, \texttt{var} - 1\},\\ \{\texttt{var} - 5, \texttt{var} - 2, \texttt{var} - 6, \texttt{var} - 2\}\end{array}\right)$$

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 $\texttt{item}_1.x > \texttt{item}_2.y$.
- We create an arc between c_i and c_{i+1}. We associate to this arc the arc constraint $\texttt{item}_1.x = \texttt{item}_2.y$.

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, immediatly 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 $\mathbf{PATH\_FROM\_TO}(\texttt{index}, 1, 0) = 1$ to $\mathbf{PATH\_FROM\_TO}(\texttt{index}, 1, 0) \geq 1$. Therefore we simplify $\underline{\overline{\mathbf{PATH\_FROM\_TO}}}$ to $\overline{\mathbf{PATH\_FROM\_TO}}$.

Automaton Figure 4.265 depicts the automaton associated to the lex_greater constraint. Let $\texttt{VAR1}_i$ and $\texttt{VAR2}_i$ respectively be the var attributes of the i^{th} items of the VECTOR1 and the VECTOR2 collections. To each pair $(\texttt{VAR1}_i, \texttt{VAR2}_i)$ corresponds a signature variable $\texttt{S}_i$ as well as the following signature constraint: $(\texttt{VAR1}_i < \texttt{VAR2}_i \Leftrightarrow \texttt{S}_i = 1) \wedge (\texttt{VAR1}_i = \texttt{VAR2}_i \Leftrightarrow \texttt{S}_i = 2) \wedge (\texttt{VAR1}_i > \texttt{VAR2}_i \Leftrightarrow \texttt{S}_i = 3)$.

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_greatereq, 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.

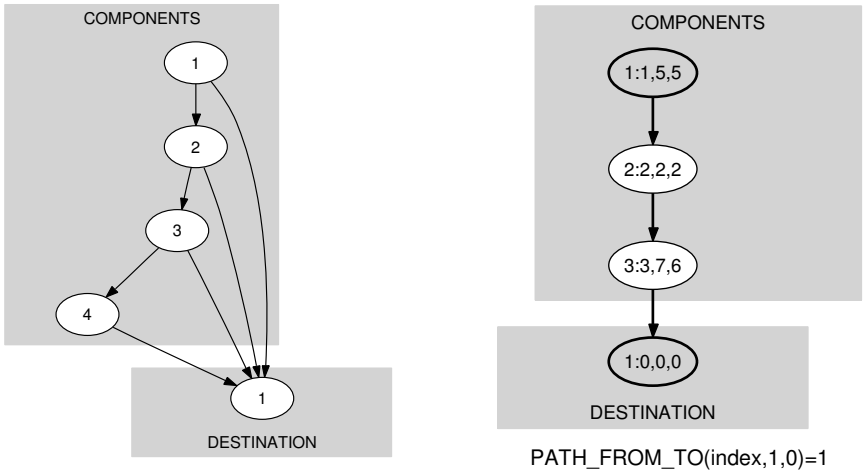


Figure 4.264: Initial and final graph of the `lex_greater` constraint

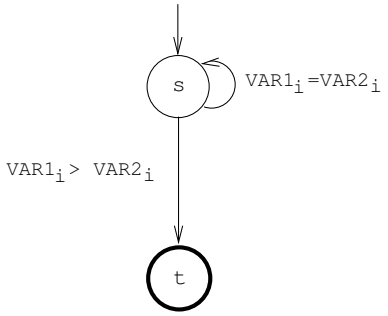


Figure 4.265: Automaton of the `lex_greater` constraint

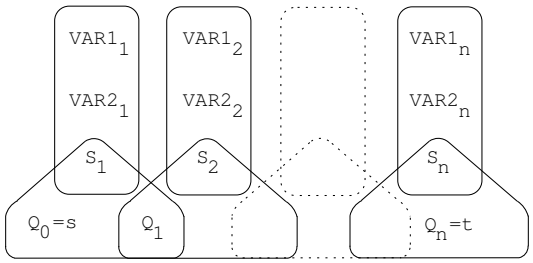


Figure 4.266: Hypergraph of the reformulation corresponding to the automaton of the `lex_greater` constraint

4.122 lex_greatereq

Origin CHIP

Constraint lex_greatereq(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 greater than or equal to VECTOR2. Given two vectors, $\vec{X}$ and $\vec{Y}$ of n components, $\langle X_0, \ldots, X_n \rangle$ and $\langle Y_0, \ldots, Y_n \rangle$, $\vec{X}$ is *lexicographically greater than or equal to* $\vec{Y}$ if and only if $n = 0$ or $X_0 > Y_0$ or $X_0 = Y_0$ and $\langle X_1, \ldots, X_n \rangle$ is lexicographically greater than or equal to $\langle Y_1, \ldots, Y_n \rangle$.

Derived Collection(s)

$$\texttt{col}\left(\begin{array}{l}\texttt{DESTINATION} - \texttt{collection}(\texttt{index} - \texttt{int}, \texttt{x} - \texttt{int}, \texttt{y} - \texttt{int}),\\ [\texttt{item}(\texttt{index} - 0, \texttt{x} - 0, \texttt{y} - 0)]\end{array}\right)$$
$$\texttt{col}\left(\begin{array}{l}\texttt{COMPONENTS} - \texttt{collection}(\texttt{index} - \texttt{int}, \texttt{x} - \texttt{dvar}, \texttt{y} - \texttt{dvar}),\\ [\texttt{item}(\texttt{index} - \texttt{VECTOR1.key}, \texttt{x} - \texttt{VECTOR1.var}, \texttt{y} - \texttt{VECTOR2.var})]\end{array}\right)$$

Arc input(s) COMPONENTS DESTINATION

Arc generator $PRODUCT(PATH, VOID) \mapsto \texttt{collection}(\texttt{item1}, \texttt{item2})$

Arc arity 2

Arc constraint(s)

$$\bigvee\left(\begin{array}{l}\texttt{item2.index} > 0 \wedge \texttt{item1.x} = \texttt{item1.y},\\ \texttt{item1.index} < |\texttt{VECTOR1}| \wedge \texttt{item2.index} = 0 \wedge \texttt{item1.x} > \texttt{item1.y},\\ \texttt{item1.index} = |\texttt{VECTOR1}| \wedge \texttt{item2.index} = 0 \wedge \texttt{item1.x} \geq \texttt{item1.y}\end{array}\right)$$

Graph property(ies) $\mathbf{PATH\_FROM\_TO}(\texttt{index}, 1, 0) = 1$

Example

$$\texttt{lex\_greatereq}\left(\begin{array}{l}\{\texttt{var} - 5, \texttt{var} - 2, \texttt{var} - 8, \texttt{var} - 9\},\\ \{\texttt{var} - 5, \texttt{var} - 2, \texttt{var} - 6, \texttt{var} - 2\}\end{array}\right)$$
$$\texttt{lex\_greatereq}\left(\begin{array}{l}\{\texttt{var} - 5, \texttt{var} - 2, \texttt{var} - 3, \texttt{var} - 9\},\\ \{\texttt{var} - 5, \texttt{var} - 2, \texttt{var} - 3, \texttt{var} - 9\}\end{array}\right)$$

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 $\texttt{item}_1.x \geq \texttt{item}_2.y$; Otherwise we associate to this arc the arc constraint $\texttt{item}_1.x > \texttt{item}_2.y$;
- We create an arc between c_i and c_{i+1}. We associate to this arc the arc constraint $\texttt{item}_1.x = \texttt{item}_2.y$.

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 $\mathbf{PATH\_FROM\_TO}(\texttt{index}, 1, 0) = 1$ to $\mathbf{PATH\_FROM\_TO}(\texttt{index}, 1, 0) \geq 1$. Therefore we simplify $\underline{\mathbf{PATH\_FROM\_TO}}$ to $\overline{\mathbf{PATH\_FROM\_TO}}$.

Automaton Figure 4.268 depicts the automaton associated to the `lex_greatereq` constraint. Let $\texttt{VAR1}_i$ and $\texttt{VAR2}_i$ respectively be the `var` attributes of the i^{th} items of the `VECTOR1` and the `VECTOR2` collections. To each pair $(\texttt{VAR1}_i, \texttt{VAR2}_i)$ corresponds a signature variable $\texttt{S}_i$ as well as the following signature constraint: $(\texttt{VAR1}_i < \texttt{VAR2}_i \Leftrightarrow \texttt{S}_i = 1) \wedge (\texttt{VAR1}_i = \texttt{VAR2}_i \Leftrightarrow \texttt{S}_i = 2) \wedge (\texttt{VAR1}_i > \texttt{VAR2}_i \Leftrightarrow \texttt{S}_i = 3)$.

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.

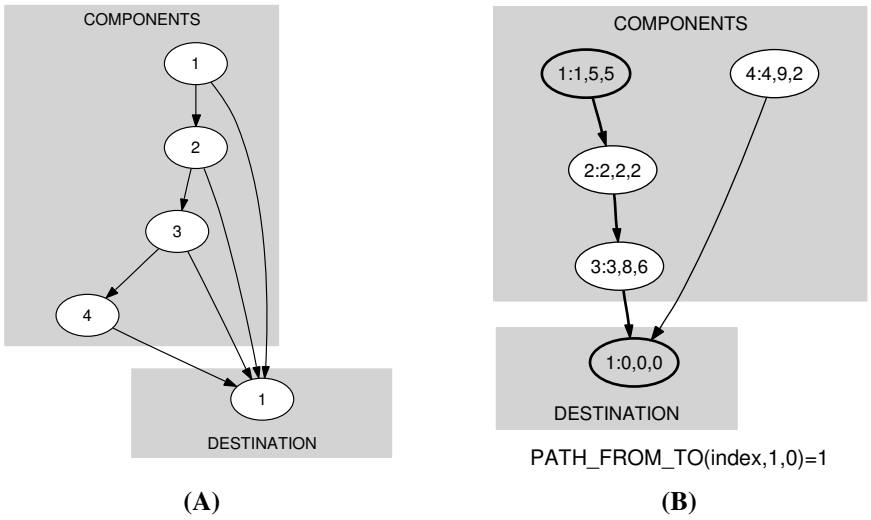


Figure 4.267: Initial and final graph of the lex_greatereq constraint

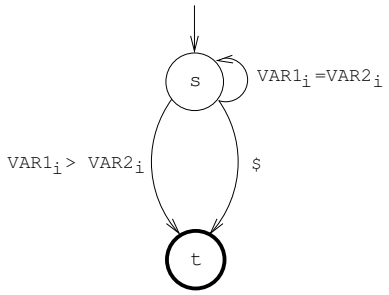


Figure 4.268: Automaton of the lex_greatereq constraint

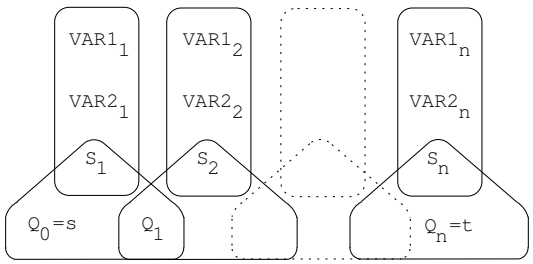


Figure 4.269: Hypergraph of the reformulation corresponding to the automaton of the lex_greatereq constraint

4.123 lex_less

Origin CHIP

Constraint lex_less(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 less than VECTOR2. Given two vectors, $\vec{X}$ and $\vec{Y}$ of n components, $\langle X_0, \ldots, X_n \rangle$ and $\langle Y_0, \ldots, Y_n \rangle$, $\vec{X}$ is *lexicographically strictly less than* $\vec{Y}$ if and only if $X_0 < Y_0$ or $X_0 = Y_0$ and $\langle X_1, \ldots, X_n \rangle$ is lexicographically strictly less than $\langle Y_1, \ldots, Y_n \rangle$.

Derived Collection(s)

$$\texttt{col}\left(\begin{array}{l}\texttt{DESTINATION} - \texttt{collection}(\texttt{index} - \texttt{int}, \texttt{x} - \texttt{int}, \texttt{y} - \texttt{int}),\\ [\texttt{item}(\texttt{index} - 0, \texttt{x} - 0, \texttt{y} - 0)]\end{array}\right)$$

$$\texttt{col}\left(\begin{array}{l}\texttt{COMPONENTS} - \texttt{collection}(\texttt{index} - \texttt{int}, \texttt{x} - \texttt{dvar}, \texttt{y} - \texttt{dvar}),\\ [\texttt{item}(\texttt{index} - \texttt{VECTOR1.key}, \texttt{x} - \texttt{VECTOR1.var}, \texttt{y} - \texttt{VECTOR2.var})]\end{array}\right)$$

Arc input(s) COMPONENTS DESTINATION

Arc generator $PRODUCT(PATH, VOID) \mapsto \texttt{collection}(\texttt{item1}, \texttt{item2})$

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) $\mathbf{PATH\_FROM\_TO}(\texttt{index}, 1, 0) = 1$

Example

$$\texttt{lex\_less}\left(\begin{array}{l}\{\texttt{var} - 5, \texttt{var} - 2, \texttt{var} - 3, \texttt{var} - 9\},\\ \{\texttt{var} - 5, \texttt{var} - 2, \texttt{var} - 6, \texttt{var} - 2\}\end{array}\right)$$

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:

- 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 $\texttt{item}_1.x < \texttt{item}_2.y$.
- We create an arc between c_i and c_{i+1}. We associate to this arc the arc constraint $\texttt{item}_1.x = \texttt{item}_2.y$.

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 **PATH_FROM_TO** is equal to 1 we can rewrite $\mathbf{PATH\_FROM\_TO}(\texttt{index}, 1, 0) = 1$ to $\mathbf{PATH\_FROM\_TO}(\texttt{index}, 1, 0) \geq 1$. Therefore we simplify $\overline{\mathbf{PATH\_FROM\_TO}}$ to $\overline{\mathbf{PATH\_FROM\_TO}}$.

Automaton Figure 4.271 depicts the automaton associated to the lex_less constraint. Let $\texttt{VAR1}_i$ and $\texttt{VAR2}_i$ respectively be the var attributes of the i^{th} items of the VECTOR1 and the VECTOR2 collections. To each pair $(\texttt{VAR1}_i, \texttt{VAR2}_i)$ corresponds a signature variable $\texttt{S}_i$ as well as the following signature constraint: $(\texttt{VAR1}_i < \texttt{VAR2}_i \Leftrightarrow \texttt{S}_i = 1) \wedge (\texttt{VAR1}_i = \texttt{VAR2}_i \Leftrightarrow \texttt{S}_i = 2) \wedge (\texttt{VAR1}_i > \texttt{VAR2}_i \Leftrightarrow \texttt{S}_i = 3)$.

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_chain_less.

See also lex_between, lex_lesseq, lex_greater, lex_greatereq, 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.

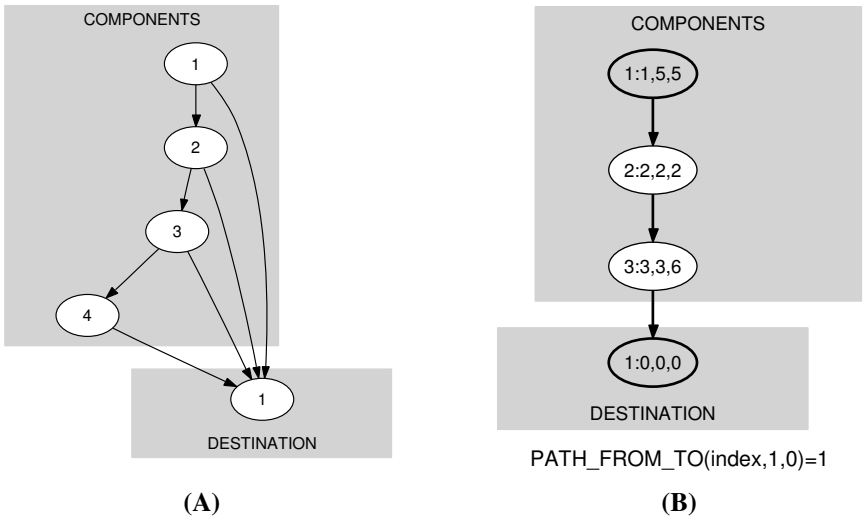


Figure 4.270: Initial and final graph of the `lex_less` constraint

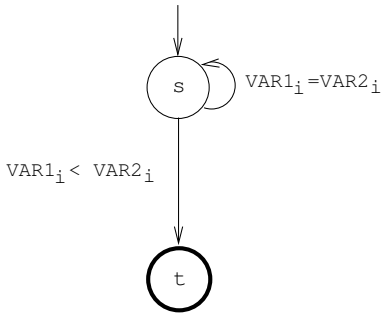


Figure 4.271: Automaton of the `lex_less` constraint

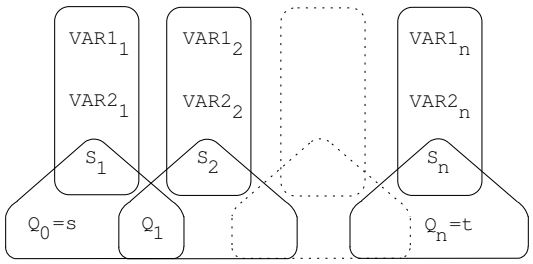


Figure 4.272: Hypergraph of the reformulation corresponding to the automaton of the `lex_less` constraint

4.124 lex_lesseq

Origin CHIP

Constraint lex_lesseq(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 less than or equal to VECTOR2. Given two vectors, $\vec{X}$ and $\vec{Y}$ of n components, $\langle X_0, \ldots, X_n \rangle$ and $\langle Y_0, \ldots, Y_n \rangle$, $\vec{X}$ 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 $\langle X_1, \ldots, X_n \rangle$ is lexicographically less than or equal to $\langle Y_1, \ldots, Y_n \rangle$.

Derived Collection(s)

$$\texttt{col}\left(\begin{array}{l}\texttt{DESTINATION} - \texttt{collection}(\texttt{index} - \texttt{int}, \texttt{x} - \texttt{int}, \texttt{y} - \texttt{int}),\\ [\texttt{item}(\texttt{index} - 0, \texttt{x} - 0, \texttt{y} - 0)]\end{array}\right)$$
$$\texttt{col}\left(\begin{array}{l}\texttt{COMPONENTS} - \texttt{collection}(\texttt{index} - \texttt{int}, \texttt{x} - \texttt{dvar}, \texttt{y} - \texttt{dvar}),\\ [\texttt{item}(\texttt{index} - \texttt{VECTOR1.key}, \texttt{x} - \texttt{VECTOR1.var}, \texttt{y} - \texttt{VECTOR2.var})]\end{array}\right)$$

Arc input(s) COMPONENTS DESTINATION

Arc generator $PRODUCT(PATH, VOID) \mapsto \texttt{collection}(\texttt{item1}, \texttt{item2})$

Arc arity 2

Arc constraint(s)

$$\bigvee\left(\begin{array}{l}\texttt{item2.index} > 0 \wedge \texttt{item1.x} = \texttt{item1.y},\\ \texttt{item1.index} < |\texttt{VECTOR1}| \wedge \texttt{item2.index} = 0 \wedge \texttt{item1.x} < \texttt{item1.y},\\ \texttt{item1.index} = |\texttt{VECTOR1}| \wedge \texttt{item2.index} = 0 \wedge \texttt{item1.x} \leq \texttt{item1.y}\end{array}\right)$$

Graph property(ies) $\mathbf{PATH\_FROM\_TO}(\texttt{index}, 1, 0) = 1$

Example

$$\texttt{lex\_lesseq}\left(\begin{array}{l}\{\texttt{var} - 5, \texttt{var} - 2, \texttt{var} - 3, \texttt{var} - 1\},\\ \{\texttt{var} - 5, \texttt{var} - 2, \texttt{var} - 6, \texttt{var} - 2\}\end{array}\right)$$
$$\texttt{lex\_lesseq}\left(\begin{array}{l}\{\texttt{var} - 5, \texttt{var} - 2, \texttt{var} - 3, \texttt{var} - 9\},\\ \{\texttt{var} - 5, \texttt{var} - 2, \texttt{var} - 3, \texttt{var} - 9\}\end{array}\right)$$

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 $\texttt{item}_1.x \leq \texttt{item}_2.y$; Otherwise we associate to this arc the arc constraint $\texttt{item}_1.x < \texttt{item}_2.y$;
- We create an arc between c_i and c_{i+1}. We associate to this arc the arc constraint $\texttt{item}_1.x = \texttt{item}_2.y$.

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 $\mathbf{PATH\_FROM\_TO}(\texttt{index}, 1, 0) = 1$ to $\mathbf{PATH\_FROM\_TO}(\texttt{index}, 1, 0) \geq 1$. Therefore we simplify $\underline{\mathbf{PATH\_FROM\_TO}}$ to $\overline{\mathbf{PATH\_FROM\_TO}}$.

Automaton Figure 4.274 depicts the automaton associated to the `lex_lesseq` constraint. Let $\texttt{VAR1}_i$ and $\texttt{VAR2}_i$ respectively be the `var` attributes of the i^{th} items of the `VECTOR1` and the `VECTOR2` collections. To each pair $(\texttt{VAR1}_i, \texttt{VAR2}_i)$ corresponds a signature variable $\texttt{S}_i$ as well as the following signature constraint: $(\texttt{VAR1}_i < \texttt{VAR2}_i \Leftrightarrow \texttt{S}_i = 1) \wedge (\texttt{VAR1}_i = \texttt{VAR2}_i \Leftrightarrow \texttt{S}_i = 2) \wedge (\texttt{VAR1}_i > \texttt{VAR2}_i \Leftrightarrow \texttt{S}_i = 3)$.

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.

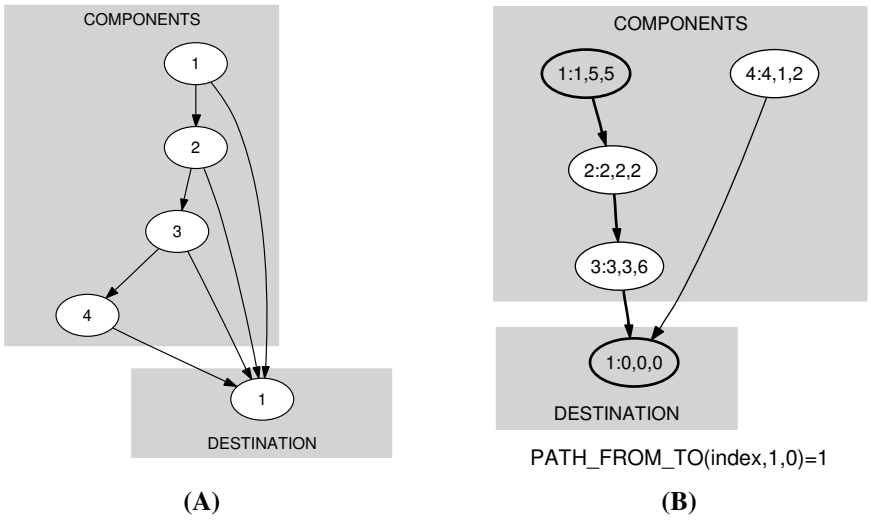


Figure 4.273: Initial and final graph of the `lex_lesseq` constraint

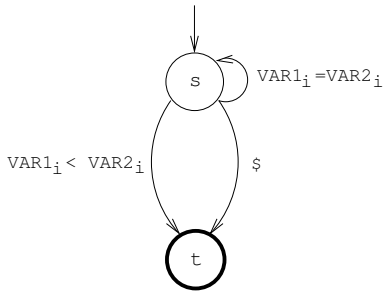


Figure 4.274: Automaton of the `lex_lesseq` constraint

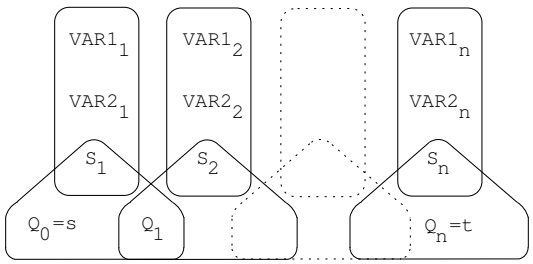


Figure 4.275: Hypergraph of the reformulation corresponding to the automaton of the `lex_lesseq` constraint

4.125 link_set_to_booleans

Origin Inspired by domain_constraint.

Constraint link_set_to_booleans(SVAR, BOOLEANS)

Argument(s)

SVAR	:	svar
BOOLEANS	:	collection(bool − dvar, val − int)

Restriction(s)
required(BOOLEANS, [bool, val])
BOOLEANS.bool ≥ 0
BOOLEANS.bool ≤ 1
distinct(BOOLEANS, val)

Purpose Make the link between a set variable SVAR and those 0-1 variables that are associated to each potential value belonging to SVAR: The 0-1 variables, which are associated to a value belonging to the set variable SVAR, are equal to 1, while the remaining 0-1 variables are all equal to 0.

Derived Collection(s) col(SET − collection(one − int, setvar − svar), [item(one − 1, setvar − SVAR)])

Arc input(s) SET BOOLEANS

Arc generator $PRODUCT \mapsto$ collection(set, booleans)

Arc arity 2

Arc constraint(s) booleans.bool = set.one ⇔ in_set(booleans.val, set.setvar)

Graph property(ies) **NARC** = |BOOLEANS|

Example

$$\texttt{link\_set\_to\_booleans}\left(\begin{array}{l}\{1,3,4\},\\ \left\{\begin{array}{ll}\texttt{bool}-0 & \texttt{val}-0,\\ \texttt{bool}-1 & \texttt{val}-1,\\ \texttt{bool}-0 & \texttt{val}-2,\\ \texttt{bool}-1 & \texttt{val}-3,\\ \texttt{bool}-1 & \texttt{val}-4,\\ \texttt{bool}-0 & \texttt{val}-5\end{array}\right\}\end{array}\right)$$

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 link_set_to_booleans 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 BOOLEANS. The arc constraint between the set variable SVAR 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 |BOOLEANS| arcs.

Signature Since the initial graph contains |BOOLEANS| arcs the maximum number of arcs of the final graph is equal to |BOOLEANS|. Therefore we can rewrite the graph property $\mathbf{NARC} = |\texttt{BOOLEANS}|$ to $\mathbf{NARC} \geq |\texttt{BOOLEANS}|$ and simplify $\underline{\mathbf{NARC}}$ to $\overline{\mathbf{NARC}}$.

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.

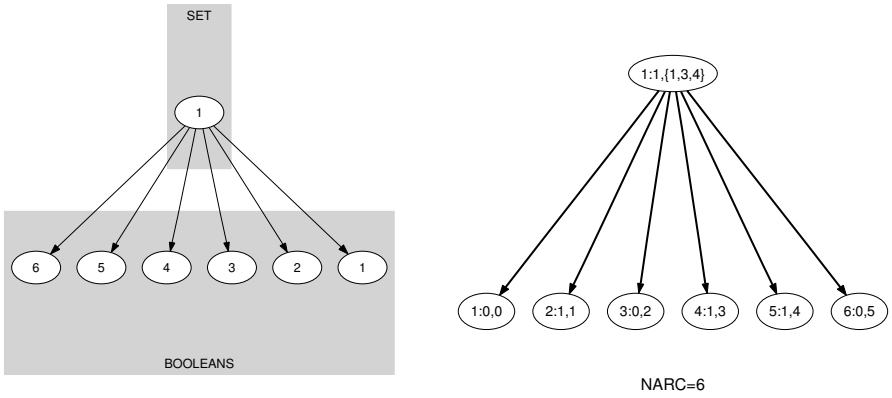


Figure 4.276: Initial and final graph of the `link_set_to_booleans` constraint

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 $\in [=, \neq, <, \geq, >, \leq]$

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 CTR Y holds; X and Y are two consecutive variables of the collection VARIABLES.

Arc input(s) VARIABLES

Arc generator $PATH \mapsto$ collection(variables1, variables2)

Arc arity 2

Arc constraint(s) variables1.var CTR variables2.var

Graph property(ies) **MAX_NCC** = SIZE

Example

$$\texttt{longest\_change}\left(4, \left\{\begin{array}{l}\texttt{var}-8,\\ \texttt{var}-8,\\ \texttt{var}-3,\\ \texttt{var}-4,\\ \texttt{var}-1,\\ \texttt{var}-1,\\ \texttt{var}-5,\\ \texttt{var}-5,\\ \texttt{var}-2\end{array}\right\}, \neq\right)$$

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 consecutives 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.

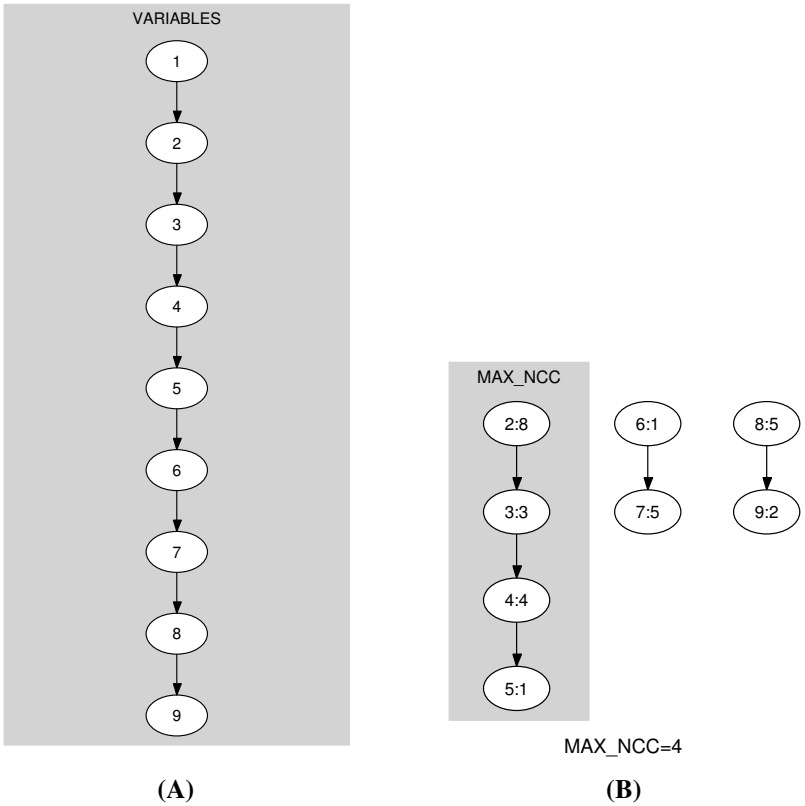


Figure 4.277: Initial and final graph of the `longest_change` constraint

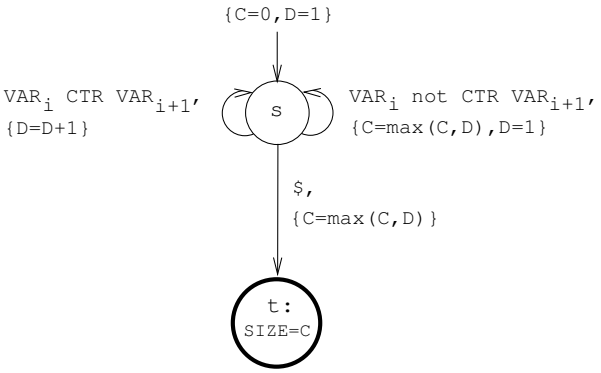


Figure 4.278: Automaton of the `longest_change` constraint

Automaton Figure 4.278 depicts the automaton associated to the `longest_change` constraint. To each pair of consecutive variables $(\texttt{VAR}_i, \texttt{VAR}_{i+1})$ of the collection `VARIABLES` corresponds a 0-1 signature variable $\texttt{S}_i$. The following signature constraint links $\texttt{VAR}_i$, $\texttt{VAR}_{i+1}$ and $\texttt{S}_i$: $\texttt{VAR}_i$ `CTR` $\texttt{VAR}_{i+1} \Leftrightarrow \texttt{S}_i$.

See also `change`.

Key words timetabling constraint, automaton, automaton with counters, sliding cyclic(1) constraint network(3).

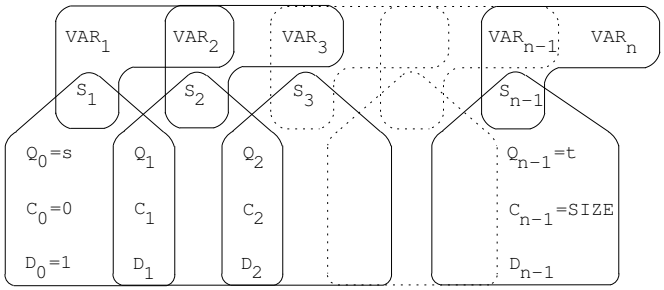


Figure 4.279: Hypergraph of the reformulation corresponding to the automaton of the `longest_change` constraint

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 $CLIQUE \mapsto$ collection(nodes1, nodes2)

Arc arity 2

Arc constraint(s) nodes1.succ = nodes2.index

Graph property(ies)

- $\mathbf{NCC}$ = NBCYCLE
- $\mathbf{NTREE}$ = NBTREE

Example

$$\texttt{map}\left(\left\{\begin{array}{ll}\texttt{index}-1 & \texttt{succ}-5,\\ \texttt{index}-2 & \texttt{succ}-9,\\ \texttt{index}-3 & \texttt{succ}-8,\\ \texttt{index}-4 & \texttt{succ}-2,\\ \texttt{index}-5 & \texttt{succ}-9,\\ \texttt{index}-6 & \texttt{succ}-2,\\ \texttt{index}-7 & \texttt{succ}-9,\\ \texttt{index}-8 & \texttt{succ}-8,\\ \texttt{index}-9 & \texttt{succ}-1\end{array}\right\}\right)$$

Parts (A) and (B) of Figure 4.280 respectively show the initial and final graph. Since we use the $\mathbf{NCC}$ 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.

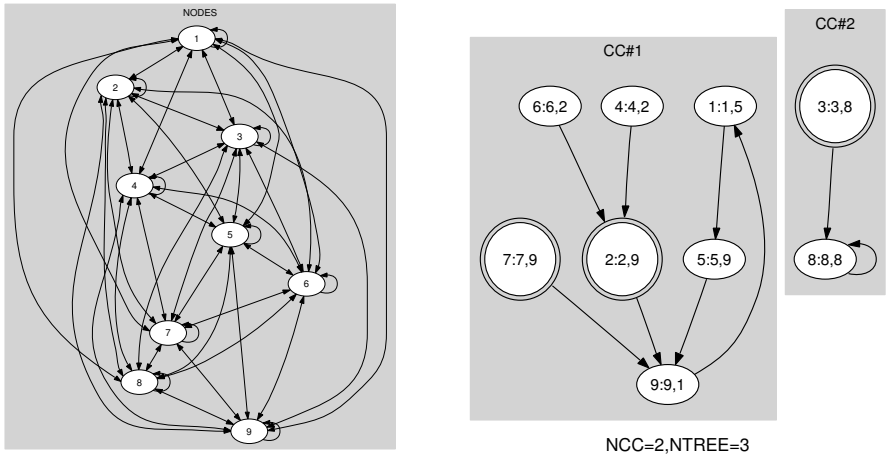


Figure 4.280: Initial and final graph of the `map` constraint

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 max_index(MAX_INDEX, VARIABLES)

Argument(s)

MAX_INDEX	:	dvar
VARIABLES	:	collection(index − int, var − dvar)

Restriction(s)

|VARIABLES| > 0
MAX_INDEX ≥ 0
MAX_INDEX ≤ |VARIABLES|
required(VARIABLES, [index, var])
VARIABLES.index ≥ 1
VARIABLES.index ≤ |VARIABLES|
distinct(VARIABLES, index)

Purpose MAX_INDEX is the index of the variables corresponding to the maximum value of the collection of variables VARIABLES.

Arc input(s) VARIABLES

Arc generator $CLIQUE \mapsto$ collection(variables1, variables2)

Arc arity 2

Arc constraint(s) variables1.key = variables2.key ∨ variables1.var > variables2.var

Graph property(ies) **ORDER**(0, 0, index) = MAX_INDEX

Example $$\texttt{max\_index}\left(3, \left\{\begin{array}{ll} \texttt{index}-1 & \texttt{var}-3,\\ \texttt{index}-2 & \texttt{var}-2,\\ \texttt{index}-3 & \texttt{var}-7,\\ \texttt{index}-4 & \texttt{var}-2,\\ \texttt{index}-5 & \texttt{var}-6 \end{array}\right\}\right)$$

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.

Automaton Figure 4.282 depicts the automaton associated to the max_index constraint. To each item of the collection VARIABLES corresponds a signature variable S_i, which is equal to 0.

See also min_index.

Key words order constraint, maximum, automaton, automaton with counters, alpha-acyclic constraint network(4).

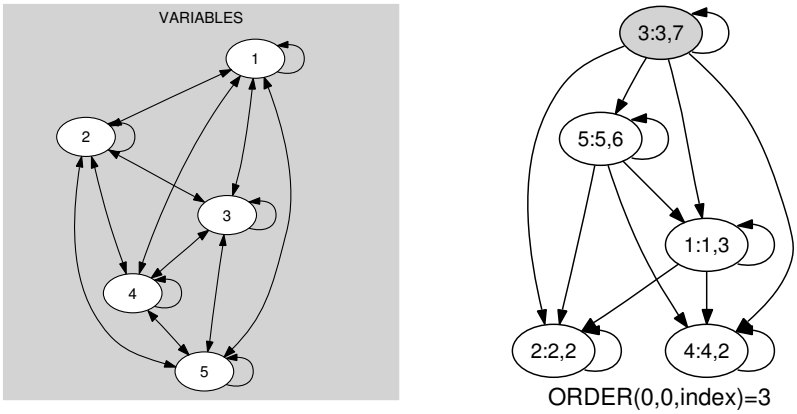


Figure 4.281: Initial and final graph of the max_index constraint

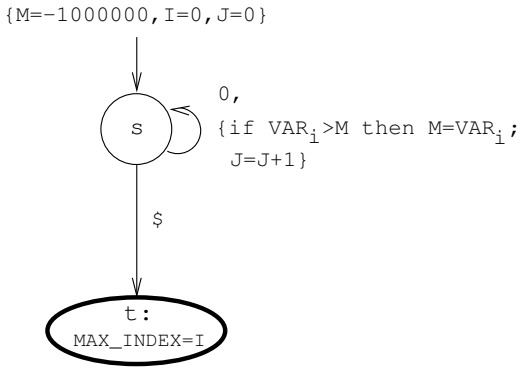


Figure 4.282: Automaton of the max_index constraint

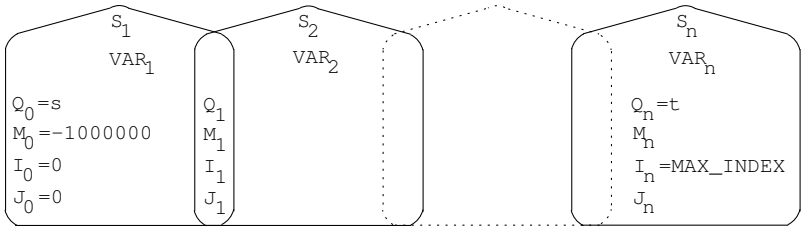


Figure 4.283: Hypergraph of the reformulation corresponding to the automaton of the max_index constraint

4.129 max_n

Origin [33]

Constraint max_n(MAX, RANK, VARIABLES)

Argument(s)

MAX	:	dvar
RANK	:	int
VARIABLES	:	collection(var − dvar)

Restriction(s)
|VARIABLES| > 0
RANK ≥ 0
RANK < |VARIABLES|
required(VARIABLES, var)

Purpose 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.

Arc input(s) VARIABLES

Arc generator $CLIQUE \mapsto$ collection(variables1, variables2)

Arc arity 2

Arc constraint(s) variables1.key = variables2.key ∨ variables1.var > variables2.var

Graph property(ies) **ORDER**(RANK, MININT, var) = MAX

Example $\texttt{max\_n}\left(\left\langle 6, 1, \left\{\begin{array}{l}\texttt{var}-3,\\ \texttt{var}-1,\\ \texttt{var}-7,\\ \texttt{var}-1,\\ \texttt{var}-6\end{array}\right\}\right\rangle\right)$

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.

Algorithm [33].

See also maximum, min_n.

Key words order constraint, rank, maximum.

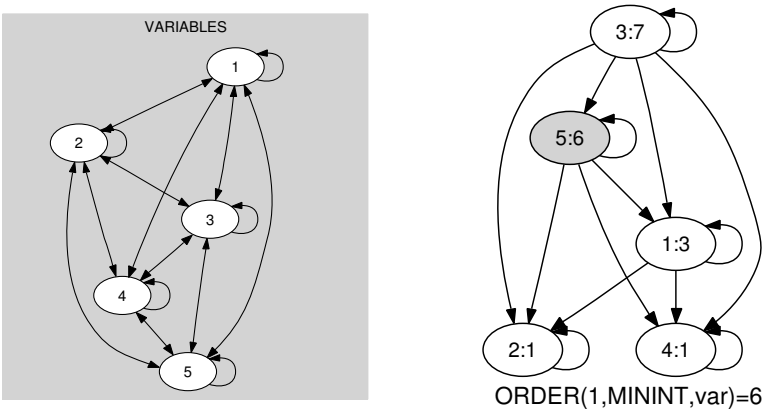


Figure 4.284: Initial and final graph of the `max_n` constraint

4.130 max_nvalue

Origin Derived from nvalue.

Constraint max_nvalue(MAX, VARIABLES)

Argument(s)

MAX	:	dvar
VARIABLES	:	collection(var − dvar)

Restriction(s)

MAX ≥ 1
MAX ≤ |VARIABLES|
required(VARIABLES, var)

Purpose MAX is the maximum number of times that the same value is taken by the variables of the collection VARIABLES.

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** = MAX

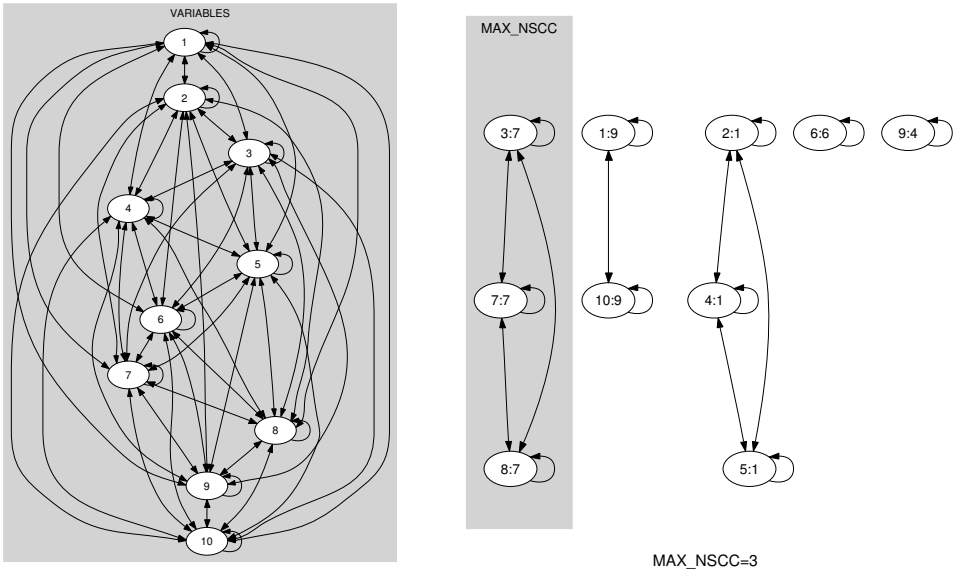
Example

$$\texttt{max\_nvalue}\left(\begin{array}{l} 3, \left\{\begin{array}{l} \texttt{var}-9,\\ \texttt{var}-1,\\ \texttt{var}-7,\\ \texttt{var}-1,\\ \texttt{var}-1,\\ \texttt{var}-6,\\ \texttt{var}-7,\\ \texttt{var}-7,\\ \texttt{var}-4,\\ \texttt{var}-9 \end{array}\right\} \end{array}\right)$$

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 VARIABLES 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 max_nvalue constraint. To each item of the collection VARIABLES corresponds a signature variable $\mathtt{S}_i$, which is equal to 0.



(A) **(B)**

Figure 4.285: Initial and final graph of the `max_nvalue` constraint

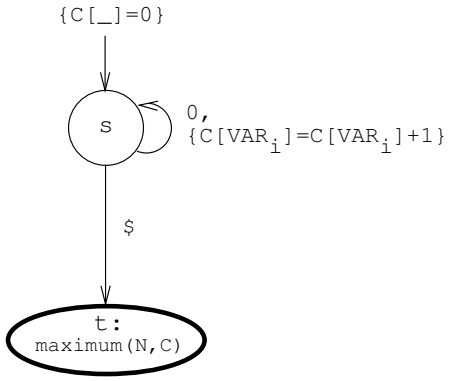


Figure 4.286: Automaton of the `max_nvalue` constraint

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.

4.131 max_size_set_of_consecutive_var

Origin N. Beldiceanu

Constraint max_size_set_of_consecutive_var(MAX, VARIABLES)

Argument(s)

MAX	:	dvar
VARIABLES	:	collection(var − dvar)

Restriction(s) MAX ≥ 1
MAX ≤ |VARIABLES|
required(VARIABLES, var)

Purpose MAX is the size of the largest set of variables of the collection VARIABLES which all take their value in a set of consecutive values.

Arc input(s) VARIABLES

Arc generator $CLIQUE \mapsto$ collection(variables1, variables2)

Arc arity 2

Arc constraint(s) abs(variables1.var − variables2.var) ≤ 1

Graph property(ies) **MAX_NSCC** = MAX

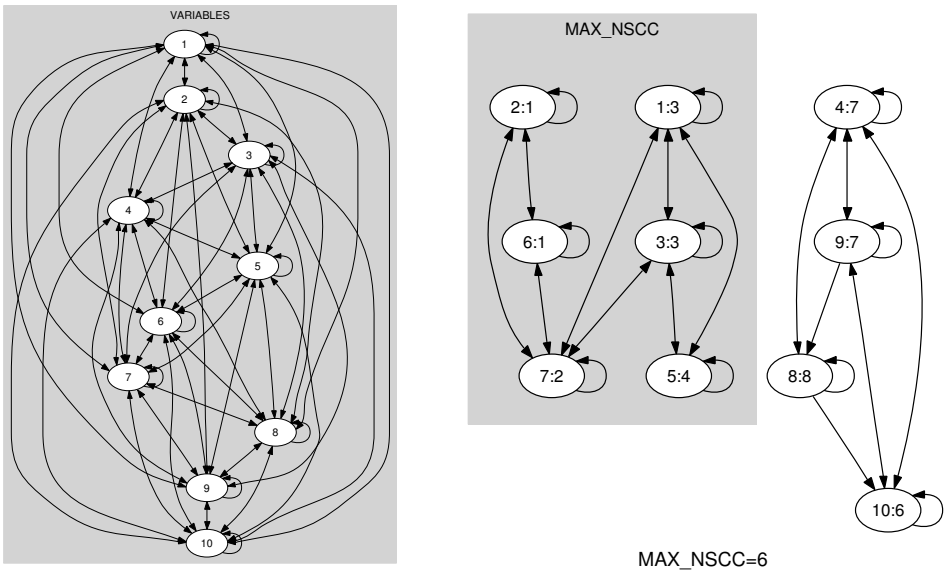
Example max_size_set_of_consecutive_var$\left(6, \left\{\begin{array}{l} \mathtt{var}-3,\\ \mathtt{var}-1,\\ \mathtt{var}-3,\\ \mathtt{var}-7,\\ \mathtt{var}-4,\\ \mathtt{var}-1,\\ \mathtt{var}-2,\\ \mathtt{var}-8,\\ \mathtt{var}-7,\\ \mathtt{var}-6 \end{array}\right\}\right)$

In the previous example, the following sets of variables $\{\mathtt{var}-3, \mathtt{var}-1, \mathtt{var}-3, \mathtt{var}-4, \mathtt{var}-1, \mathtt{var}-2\}$ and $\{\mathtt{var}-7, \mathtt{var}-8, \mathtt{var}-7, \mathtt{var}-6\}$ 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 nset_of_consecutive_values.

Key words value constraint, consecutive values, maximum.



(A) **(B)**

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.

Arc input(s) VARIABLES

Arc generator $CLIQUE \mapsto$ collection(variables1, variables2)

Arc arity 2

Arc constraint(s) variables1.key = variables2.key ∨ variables1.var > variables2.var

Graph property(ies) **ORDER**(0, MININT, var) = MAX

Example $\texttt{maximum}\left(\left\langle 7, \left\{\begin{array}{l}\texttt{var}-3,\\ \texttt{var}-2,\\ \texttt{var}-7,\\ \texttt{var}-2,\\ \texttt{var}-6\end{array}\right\}\right\rangle\right)$

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.

Graph model We use a similar definition that the one that was utilized for the minimum constraint. Within the arc constraint, we replace the comparaison operator $<$ by $>$.

Automaton Figure 4.289 depicts the automaton associated to the maximum constraint. Let $\texttt{VAR}_i$ be the i^{th} variable of the VARIABLES collection. To each pair $(\texttt{MAX}, \texttt{VAR}_i)$ corresponds a signature variable $\texttt{S}_i$ as well as the following signature constraint: $(\texttt{MAX} > \texttt{VAR}_i \Leftrightarrow \texttt{S}_i = 0) \wedge (\texttt{MAX} = \texttt{VAR}_i \Leftrightarrow \texttt{S}_i = 1) \wedge (\texttt{MAX} < \texttt{VAR}_i \Leftrightarrow \texttt{S}_i = 2)$.

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.

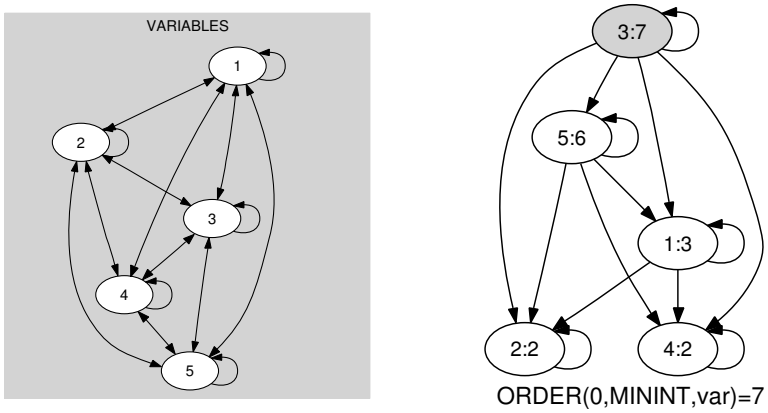


Figure 4.288: Initial and final graph of the `maximum` constraint

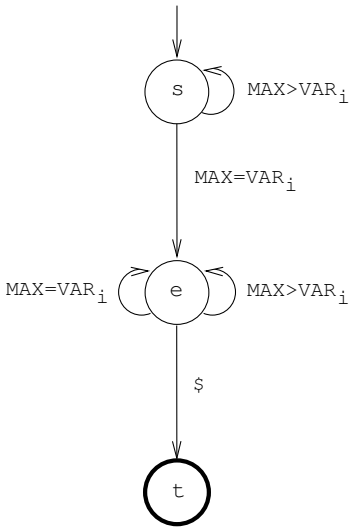


Figure 4.289: Automaton of the `maximum` constraint

Algorithm [33].

See also `minimum`.

Key words order constraint, maximum, automaton, automaton without counters, centered cyclic(1) constraint network(1).

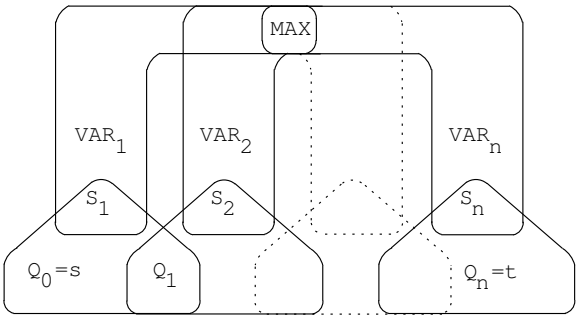


Figure 4.290: Hypergraph of the reformulation corresponding to the automaton of the `maximum` constraint

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: $(X \bmod \mathtt{M}) < (Y \bmod \mathtt{M})$.

Arc input(s) VARIABLES

Arc generator $CLIQUE \mapsto$ collection(variables1, variables2)

Arc arity 2

Arc constraint(s) variables1.key = variables2.key ∨ variables1.var mod M > variables2.var mod M

Graph property(ies) **ORDER**(0, MININT, var) = MAX

Example $$\mathtt{maximum\_modulo}\left(\begin{array}{c} 5, \left\{\begin{array}{l} \mathtt{var}-9,\\ \mathtt{var}-1,\\ \mathtt{var}-7,\\ \mathtt{var}-6,\\ \mathtt{var}-5 \end{array}\right\}, 3 \end{array}\right)$$

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.

See also maximum, minimum_modulo.

Key words order constraint, modulo, maximum.

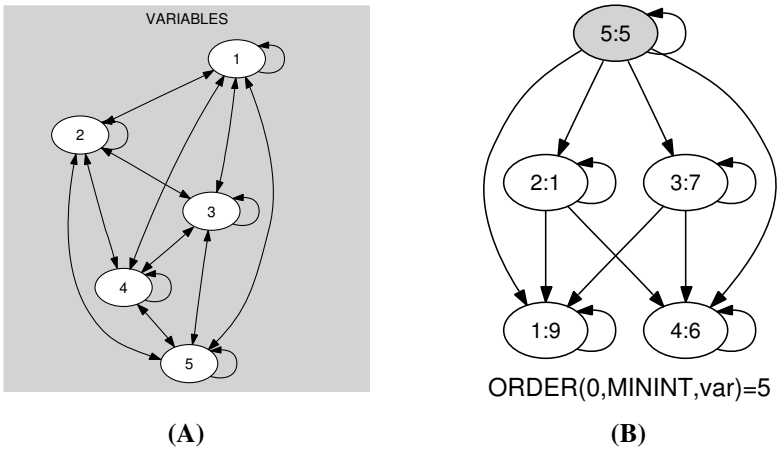


Figure 4.291: Initial and final graph of the `maximum_modulo` constraint

4.134 min_index

Origin N. Beldiceanu

Constraint min_index(MIN_INDEX, VARIABLES)

Argument(s)

MIN_INDEX	:	dvar
VARIABLES	:	collection(index − int, var − dvar)

Restriction(s)

|VARIABLES| > 0
MIN_INDEX ≥ 0
MIN_INDEX ≤ |VARIABLES|
required(VARIABLES, [index, var])
VARIABLES.index ≥ 1
VARIABLES.index ≤ |VARIABLES|
distinct(VARIABLES, index)

Purpose MIN_INDEX is the index of the variables corresponding to the minimum value of the collection of variables VARIABLES.

Arc input(s) VARIABLES

Arc generator $CLIQUE \mapsto$ collection(variables1, variables2)

Arc arity 2

Arc constraint(s) variables1.key = variables2.key ∨ variables1.var < variables2.var

Graph property(ies) **ORDER**(0, 0, index) = MIN_INDEX

Example

$$\texttt{min\_index}\left(\begin{array}{cc} 2, & \left\{\begin{array}{ll} \texttt{index}-1 & \texttt{var}-3,\\ \texttt{index}-2 & \texttt{var}-2,\\ \texttt{index}-3 & \texttt{var}-7,\\ \texttt{index}-4 & \texttt{var}-2,\\ \texttt{index}-5 & \texttt{var}-6 \end{array}\right\} \end{array}\right)$$

$$\texttt{min\_index}\left(\begin{array}{cc} 4, & \left\{\begin{array}{ll} \texttt{index}-1 & \texttt{var}-3,\\ \texttt{index}-2 & \texttt{var}-2,\\ \texttt{index}-3 & \texttt{var}-7,\\ \texttt{index}-4 & \texttt{var}-2,\\ \texttt{index}-5 & \texttt{var}-6 \end{array}\right\} \end{array}\right)$$

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 Within the context of scheduling, assume the variables of the VARIABLES collection correspond to the starts of a set of tasks. Then MIN_INDEX gives the indexes of those tasks which can be scheduled first.

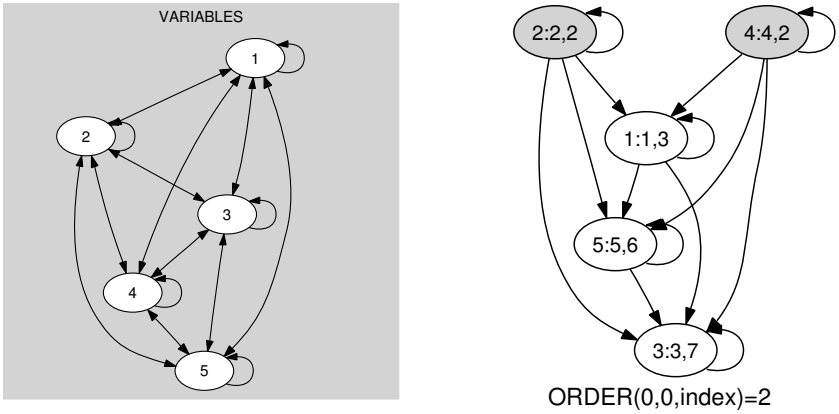


Figure 4.292: Initial and final graph of the `min_index` constraint

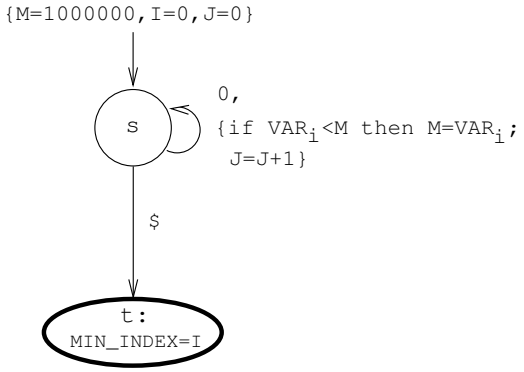


Figure 4.293: Automaton of the `min_index` constraint

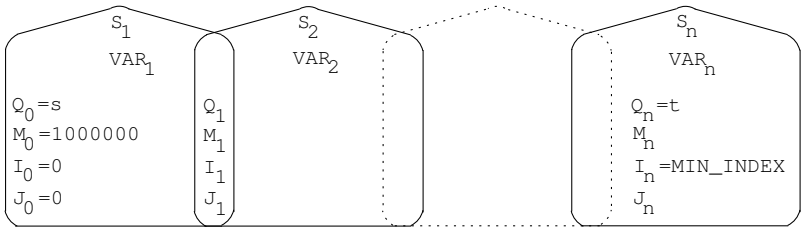


Figure 4.294: Hypergraph of the reformulation corresponding to the automaton of the `min_index` constraint

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 $\mathtt{S}_i$, which is equal to 0.

See also `max_index`.

Key words order constraint, minimum, automaton, automaton with counters, alpha-acyclic constraint network(4).

4.135 min_n

Origin [33]

Constraint min_n(MIN, RANK, VARIABLES)

Argument(s)

MIN	:	dvar
RANK	:	int
VARIABLES	:	collection(var − dvar)

Restriction(s)

|VARIABLES| > 0
RANK ≥ 0
RANK < |VARIABLES|
required(VARIABLES, var)

Purpose MIN is the minimum value of rank RANK (i.e. the RANKth smallest distinct value) of the collection of domain variables VARIABLES. Sources have a rank of 0.

Arc input(s) VARIABLES

Arc generator $CLIQUE \mapsto$ collection(variables1, variables2)

Arc arity 2

Arc constraint(s) variables1.key = variables2.key ∨ variables1.var < variables2.var

Graph property(ies) **ORDER**(RANK, MAXINT, var) = MIN

Example

$$\texttt{min\_n}\left(\left\langle 3, 1, \left\{\begin{array}{l}\texttt{var}-3,\\ \texttt{var}-1,\\ \texttt{var}-7,\\ \texttt{var}-1,\\ \texttt{var}-6\end{array}\right\}\right\rangle\right)$$

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.

Graph model A generalization of the minimum constraint.

Automaton Figure 4.296 depicts the automaton associated to the min_n constraint. Figure 4.296 depicts the automaton associated to the min_n constraint. To each item of the collection VARIABLES corresponds a signature variable S_i, which is equal to 1.

Algorithm [33].

See also minimum, max_n, ith_pos_different_from_0.

Key words order constraint, rank, minimum, maxint, automaton, automaton with array of counters.

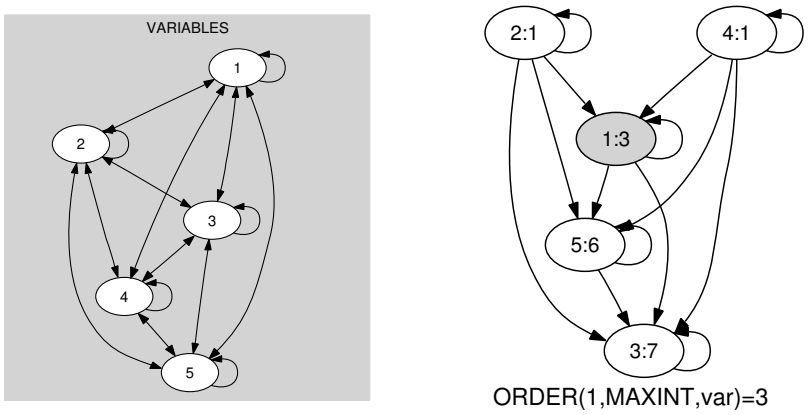


Figure 4.295: Initial and final graph of the min_n constraint

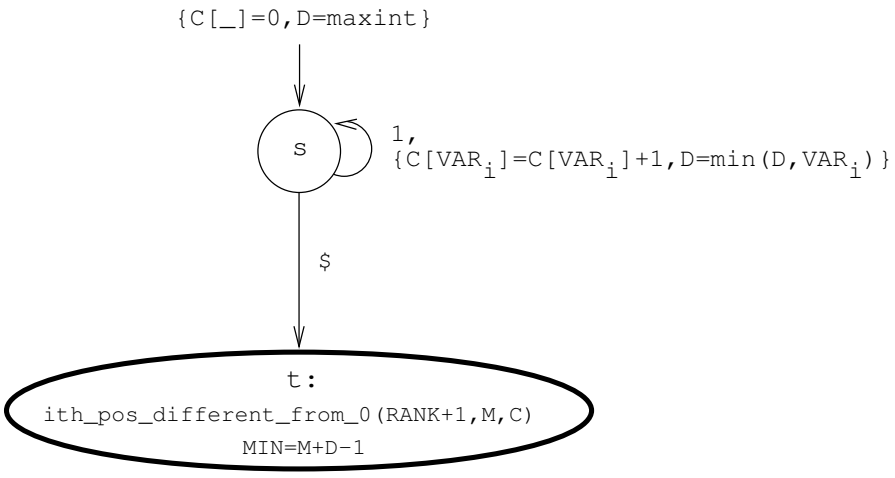


Figure 4.296: Automaton of the min_n constraint

4.136 min_nvalue

Origin N. Beldiceanu

Constraint min_nvalue(MIN, VARIABLES)

Argument(s)

MIN	:	dvar
VARIABLES	:	collection(var − dvar)

Restriction(s) MIN ≥ 1
MIN ≤ |VARIABLES|
required(VARIABLES, var)

Purpose MIN is the minimum number of times that the same value is taken by the variables of the collection VARIABLES.

Arc input(s) VARIABLES

Arc generator $CLIQUE \mapsto$ collection(variables1, variables2)

Arc arity 2

Arc constraint(s) variables1.var = variables2.var

Graph property(ies) **MIN_NSCC** = MIN

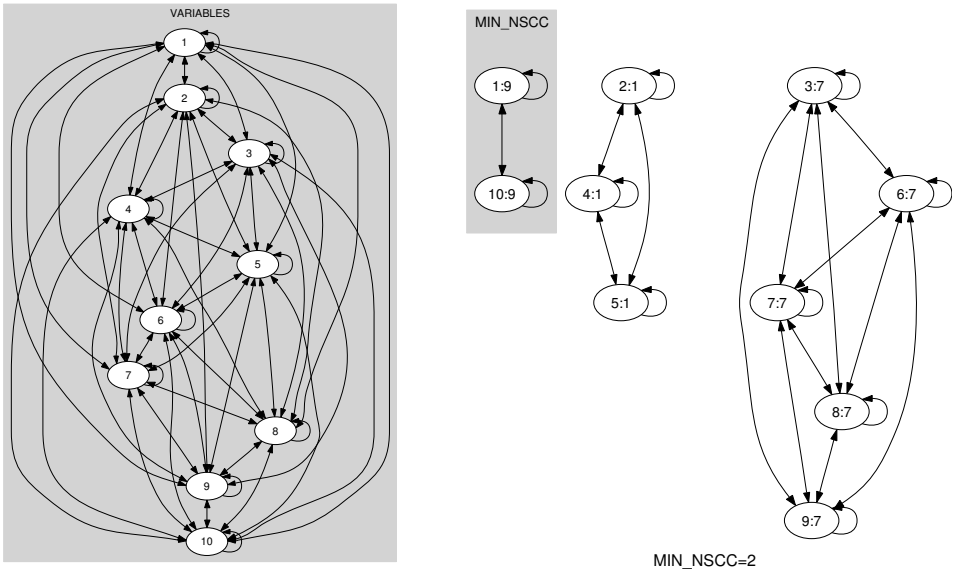
Example

$$\texttt{min\_nvalue}\left(\begin{array}{l} 2, \left\{\begin{array}{l} \texttt{var}-9,\\ \texttt{var}-1,\\ \texttt{var}-7,\\ \texttt{var}-1,\\ \texttt{var}-1,\\ \texttt{var}-7,\\ \texttt{var}-7,\\ \texttt{var}-7,\\ \texttt{var}-7,\\ \texttt{var}-9 \end{array}\right\} \end{array}\right)$$

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 **MIN_NSCC** graph property, we show the smallest strongly connected component of the final graph.

Automaton Figure 4.298 depicts the automaton associated to the min_nvalue constraint. To each item of the collection VARIABLES corresponds a signature variable $\mathtt{S}_i$, which is equal to 0.

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 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 global_cardinality constraint.



(A) **(B)**

Figure 4.297: Initial and final graph of the `min_nvalue` constraint

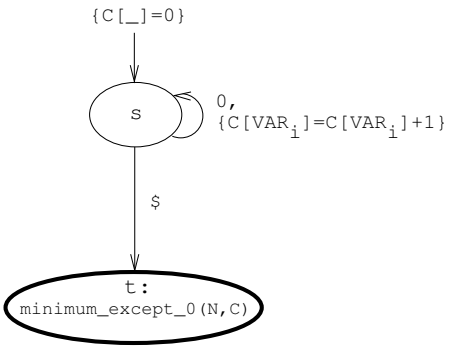


Figure 4.298: Automaton of the `min_nvalue` constraint

See also nvalue, max_nvalue.

Key words value constraint, assignment, minimum number of occurrences, minimum, automaton, automaton with array of counters, equivalence.

4.137 min_size_set_of_consecutive_var

Origin N. Beldiceanu

Constraint min_size_set_of_consecutive_var(MIN, VARIABLES)

Argument(s)

MIN	:	dvar
VARIABLES	:	collection(var − dvar)

Restriction(s)

MIN ≥ 1
MIN ≤ |VARIABLES|
required(VARIABLES, var)

Purpose MIN is the size of the smallest set of variables of the collection VARIABLES which all take their value in a set of consecutive values.

Arc input(s) VARIABLES

Arc generator $CLIQUE \mapsto$ collection(variables1, variables2)

Arc arity 2

Arc constraint(s) abs(variables1.var − variables2.var) ≤ 1

Graph property(ies) MIN_NSCC = MIN

Example min_size_set_of_consecutive_var $\left(4, \left\{\begin{array}{l} \mathtt{var}-3,\\ \mathtt{var}-1,\\ \mathtt{var}-3,\\ \mathtt{var}-7,\\ \mathtt{var}-4,\\ \mathtt{var}-1,\\ \mathtt{var}-2,\\ \mathtt{var}-8,\\ \mathtt{var}-7,\\ \mathtt{var}-6 \end{array}\right\}\right)$

In the previous example, the following sets of variables $\{\mathtt{var}-3, \mathtt{var}-1, \mathtt{var}-3, \mathtt{var}-4, \mathtt{var}-1, \mathtt{var}-2\}$ and $\{\mathtt{var}-7, \mathtt{var}-8, \mathtt{var}-7, \mathtt{var}-6\}$ take their values in the two following sets of consecutive values $\{1, 2, 3, 4\}$ and $\{6, 7, 8\}$. The min_size_set_of_consecutive_var 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 MIN_NSCC graph property, we show the smallest 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 nset_of_consecutive_values.

Key words value constraint, assignment, consecutive values, minimum.

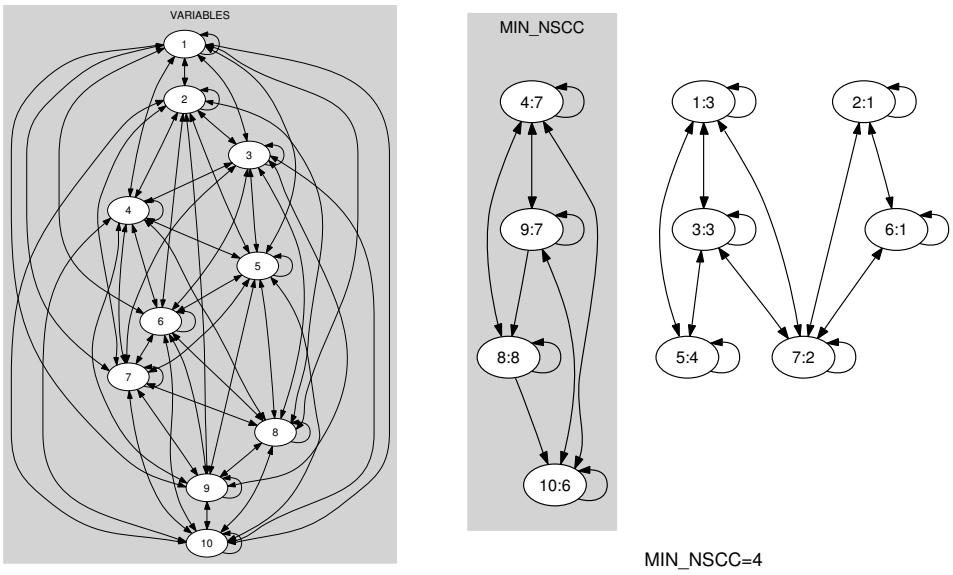


Figure 4.299: Initial and final graph of the `min_size_set_of_consecutive_var` constraint

4.138 minimum

Origin CHIP

Constraint minimum(MIN, VARIABLES)

Argument(s)

MIN	:	dvar
VARIABLES	:	collection(var − dvar)

Restriction(s) |VARIABLES| > 0
required(VARIABLES, var)

Purpose MIN is the minimum value of the collection of domain variables VARIABLES.

Arc input(s) VARIABLES

Arc generator $CLIQUE \mapsto$ collection(variables1, variables2)

Arc arity 2

Arc constraint(s) variables1.key = variables2.key ∨ variables1.var < variables2.var

Graph property(ies) **ORDER**(0, MAXINT, var) = MIN

Example $$\texttt{minimum}\left(\begin{array}{c} 2, \left\{\begin{array}{l} \texttt{var}-3, \\ \texttt{var}-2, \\ \texttt{var}-7, \\ \texttt{var}-2, \\ \texttt{var}-6 \end{array}\right\} \end{array}\right)$$

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.

Graph model The condition variables1.key = variables2.key holds if and only if variables1 and variables2 corresponds to the same vertex. It is used in order to enforce to keep all the vertices of the initial graph. **ORDER**(0, MAXINT, var) 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 minimum constraint. Let $\texttt{VAR}_i$ be the i^{th} variable of the VARIABLES collection. To each pair $(\texttt{MIN}, \texttt{VAR}_i)$ corresponds a signature variable $\texttt{S}_i$ as well as the following signature constraint: $(\texttt{MIN} < \texttt{VAR}_i \Leftrightarrow \texttt{S}_i = 0) \wedge (\texttt{MIN} = \texttt{VAR}_i \Leftrightarrow \texttt{S}_i = 1) \wedge (\texttt{MIN} > \texttt{VAR}_i \Leftrightarrow \texttt{S}_i = 2)$.

Remark Note that minimum is a constraint and not just a function that computes the minimum value of a collection of variables: The values of MIN influence the variables and reciprocally the values of the variables influence MIN.

Algorithm [33].

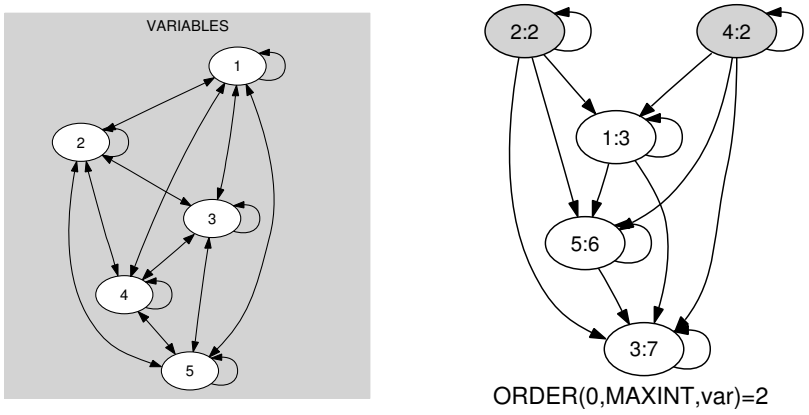


Figure 4.300: Initial and final graph of the `minimum` constraint

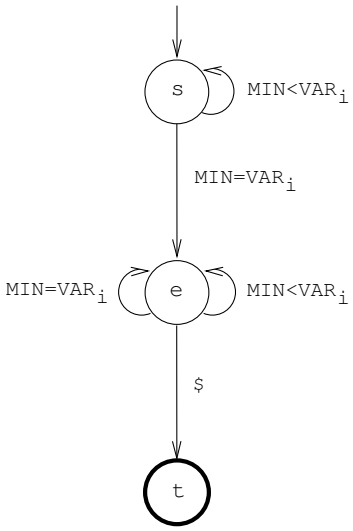


Figure 4.301: Automaton of the `minimum` constraint

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).

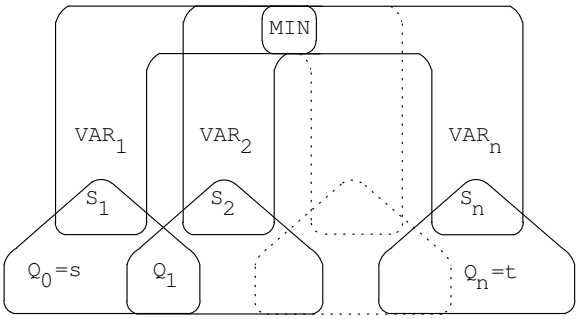


Figure 4.302: Hypergraph of the reformulation corresponding to the automaton of the minimum constraint

4.139 minimum_except_0

Origin Derived from `minimum`.

Constraint `minimum_except_0(MIN, VARIABLES)`

Argument(s)

`MIN`	:	`dvar`
`VARIABLES`	:	`collection(var − dvar)`

Restriction(s)
- $|\texttt{VARIABLES}| > 0$
- `required(VARIABLES, var)`
- $\texttt{VARIABLES.var} \geq 0$

Purpose `MIN` is the minimum value of the collection of domain variables `VARIABLES`, ignoring all variables that take 0 as value.

Arc input(s) `VARIABLES`

Arc generator $CLIQUE \mapsto \texttt{collection(variables1, variables2)}$

Arc arity 2

Arc constraint(s)
- $\texttt{variables1.var} \neq 0$
- $\texttt{variables2.var} \neq 0$
- $\texttt{variables1.key} = \texttt{variables2.key} \vee \texttt{variables1.var} < \texttt{variables2.var}$

Graph property(ies) $\mathbf{ORDER}(0, \texttt{MAXINT}, \texttt{var}) = \texttt{MIN}$

Example

$$\texttt{minimum\_except\_0}\left(\left\langle 3, \left\{\begin{array}{l}\texttt{var}-3,\\ \texttt{var}-7,\\ \texttt{var}-6,\\ \texttt{var}-7,\\ \texttt{var}-4,\\ \texttt{var}-7\end{array}\right\}\right\rangle\right)$$

$$\texttt{minimum\_except\_0}\left(\left\langle 2, \left\{\begin{array}{l}\texttt{var}-3,\\ \texttt{var}-2,\\ \texttt{var}-0,\\ \texttt{var}-7,\\ \texttt{var}-2,\\ \texttt{var}-6\end{array}\right\}\right\rangle\right)$$

$$\texttt{minimum\_except\_0}\left(\left\langle 1000000, \left\{\begin{array}{l}\texttt{var}-0,\\ \texttt{var}-0,\\ \texttt{var}-0,\\ \texttt{var}-0,\\ \texttt{var}-0,\\ \texttt{var}-0\end{array}\right\}\right\rangle\right)$$

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 $\mathtt{VAR}_i$ be the i^{th} variable of the VARIABLES collection. To each pair $(\mathtt{MIN}, \mathtt{VAR}_i)$ corresponds a signature variable $\mathtt{S}_i$ as well as the following signature constraint:

$$((\mathtt{VAR}_i = 0) \wedge (\mathtt{MIN} \neq \mathtt{MAXINT})) \Leftrightarrow \mathtt{S}_i = 0 \wedge$$
$$((\mathtt{VAR}_i = 0) \wedge (\mathtt{MIN} = \mathtt{MAXINT})) \Leftrightarrow \mathtt{S}_i = 1 \wedge$$
$$((\mathtt{VAR}_i \neq 0) \wedge (\mathtt{MIN} = \mathtt{VAR}_i)) \Leftrightarrow \mathtt{S}_i = 2 \wedge$$
$$((\mathtt{VAR}_i \neq 0) \wedge (\mathtt{MIN} < \mathtt{VAR}_i)) \Leftrightarrow \mathtt{S}_i = 3.$$

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).

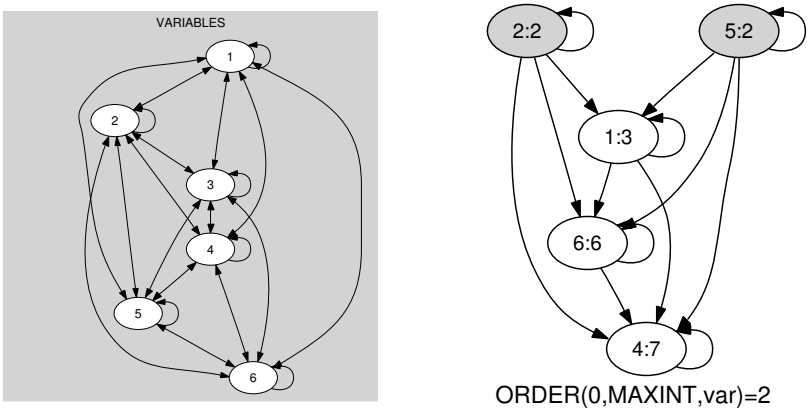


Figure 4.303: Initial and final graph of the `minimum_except_0` constraint

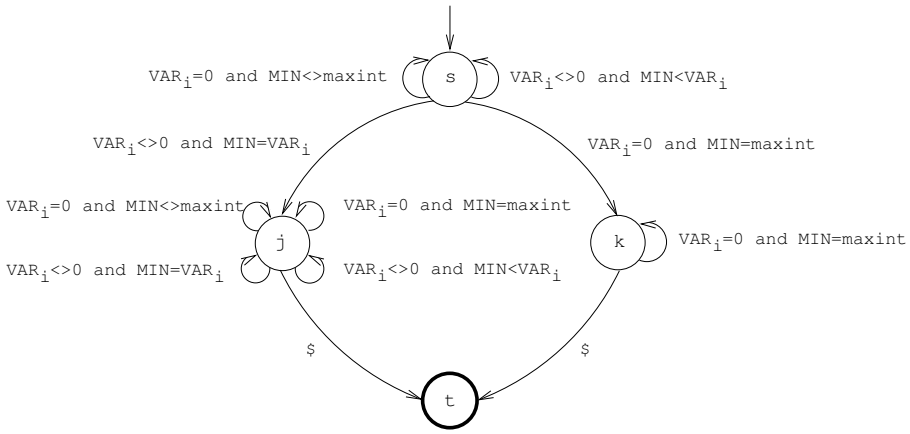


Figure 4.304: Automaton of the `minimum_except_0` constraint

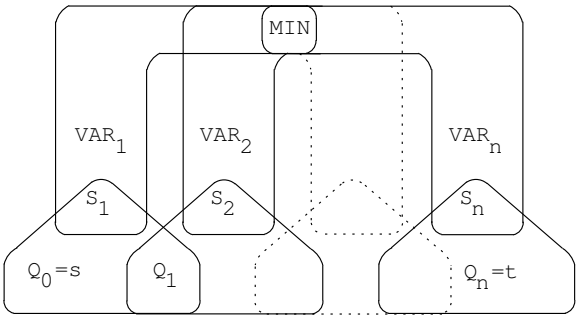


Figure 4.305: Hypergraph of the reformulation corresponding to the automaton of the `minimum_except_0` constraint

4.140 minimum_greater_than

Origin N. Beldiceanu

Constraint minimum_greater_than(VAR1, VAR2, VARIABLES)

Argument(s)

VAR1	:	dvar
VAR2	:	dvar
VARIABLES	:	collection(var − dvar)

Restriction(s) |VARIABLES| > 0
required(VARIABLES, var)

Purpose VAR1 is the smallest value strictly greater than VAR2 of the collection of variables VARIABLES: This concretely means that there exist at least one variable of VARIABLES which take a value strictly greater than VAR1.

Derived Collection(s) col(ITEM − collection(var − dvar), [item(var − VAR2)])

Arc input(s) ITEM VARIABLES

Arc generator $PRODUCT \mapsto$ collection(item, variables)

Arc arity 2

Arc constraint(s) item.var < variables.var

Graph property(ies) $\mathbf{NARC} > 0$

Sets SUCC $\mapsto$ [source, variables]

Constraint(s) on sets minimum(VAR1, variables)

Example minimum_greater_than$\left(5, 3, \left\{\begin{array}{l}\text{var} - 8,\\ \text{var} - 5,\\ \text{var} - 3,\\ \text{var} - 8\end{array}\right\}\right)$

The minimum_greater_than 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 $\mathbf{NARC}$ 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 VAR2 and to the variables of the VARIABLES collection which are strictly greater than VAR2. VAR1 is set to the smallest value of the var attribute of the sinks of the final graph.

Graph model Similar to the next_greater_element constraint, except that there is no order on the variables of the collection VARIABLES.

Automaton

Figure 4.307 depicts the automaton associated to the `minimum_greater_than` constraint. Let $\texttt{VAR}_i$ be the i^{th} variable of the `VARIABLES` collection. To each triple $(\texttt{VAR1}, \texttt{VAR2}, \texttt{VAR}_i)$ corresponds a signature variable $\texttt{S}_i$ as well as the following signature constraint:

$$((\texttt{VAR}_i < \texttt{VAR1}) \wedge (\texttt{VAR}_i \leq \texttt{VAR2})) \Leftrightarrow \texttt{S}_i = 0 \wedge$$
$$((\texttt{VAR}_i = \texttt{VAR1}) \wedge (\texttt{VAR}_i \leq \texttt{VAR2})) \Leftrightarrow \texttt{S}_i = 1 \wedge$$
$$((\texttt{VAR}_i > \texttt{VAR1}) \wedge (\texttt{VAR}_i \leq \texttt{VAR2})) \Leftrightarrow \texttt{S}_i = 2 \wedge$$
$$((\texttt{VAR}_i < \texttt{VAR1}) \wedge (\texttt{VAR}_i > \texttt{VAR2})) \Leftrightarrow \texttt{S}_i = 3 \wedge$$
$$((\texttt{VAR}_i = \texttt{VAR1}) \wedge (\texttt{VAR}_i > \texttt{VAR2})) \Leftrightarrow \texttt{S}_i = 4 \wedge$$
$$((\texttt{VAR}_i > \texttt{VAR1}) \wedge (\texttt{VAR}_i > \texttt{VAR2})) \Leftrightarrow \texttt{S}_i = 5.$$

The automaton is constructed in order to fullfit the following conditions:

- We look for an item of the `VARIABLES` collection such that $\texttt{var}_i = \texttt{VAR1}$ and $\texttt{var}_i > \texttt{VAR2}$,
- There should not exist any item of the `VARIABLES` collection such that $\texttt{var}_i < \texttt{VAR1}$ and $\texttt{var}_i > \texttt{VAR2}$.

See also

`next_greater_element`.

Key words

order constraint, minimum, automaton, automaton without counters, centered cyclic(2) constraint network(1), derived collection.

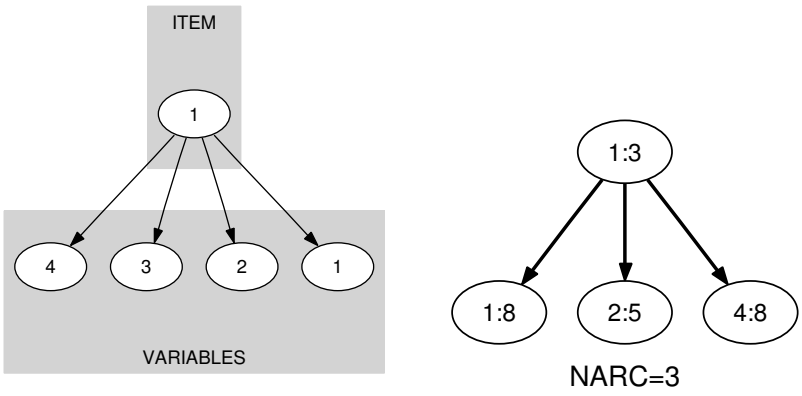


Figure 4.306: Initial and final graph of the `minimum_greater_than` constraint

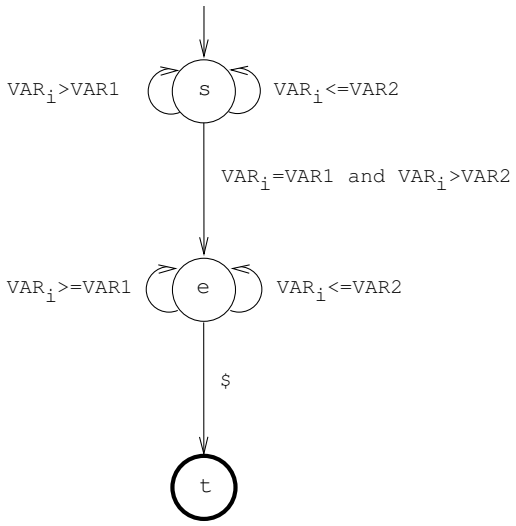


Figure 4.307: Automaton of the `minimum_greater_than` constraint

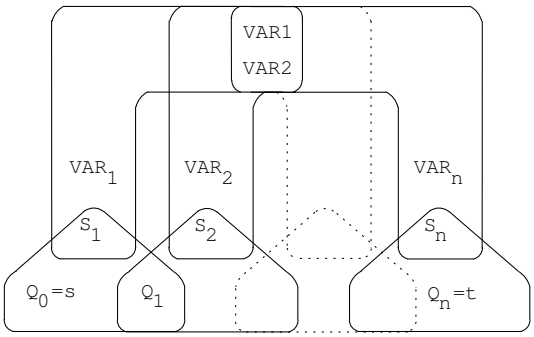


Figure 4.308: Hypergraph of the reformulation corresponding to the automaton of the `minimum_greater_than` constraint

4.141 minimum_modulo

Origin Derived from `minimum`.

Constraint `minimum_modulo(MIN, VARIABLES, M)`

Argument(s)

`MIN`	:	`dvar`
`VARIABLES`	:	`collection(var − dvar)`
`M`	:	`int`

Restriction(s) $|\texttt{VARIABLES}| > 0$
$\texttt{M} > 0$
`required(VARIABLES, var)`

Purpose `MIN` is a minimum value of the collection of domain variables `VARIABLES` according to the following partial ordering: $(X \bmod \texttt{M}) < (Y \bmod \texttt{M})$.

Arc input(s) `VARIABLES`

Arc generator $CLIQUE \mapsto \texttt{collection(variables1, variables2)}$

Arc arity 2

Arc constraint(s) $\texttt{variables1.key} = \texttt{variables2.key} \vee \texttt{variables1.var} \bmod \texttt{M} < \texttt{variables2.var} \bmod \texttt{M}$

Graph property(ies) $\mathbf{ORDER}(0, \texttt{MAXINT}, \texttt{var}) = \texttt{MIN}$

Example

$$\texttt{minimum\_modulo}\left(\begin{array}{l} 6, \left\{\begin{array}{l}\texttt{var}-9,\\ \texttt{var}-1,\\ \texttt{var}-7,\\ \texttt{var}-6,\\ \texttt{var}-5\end{array}\right\}, 3\end{array}\right)$$

$$\texttt{minimum\_modulo}\left(\begin{array}{l} 9, \left\{\begin{array}{l}\texttt{var}-9,\\ \texttt{var}-1,\\ \texttt{var}-7,\\ \texttt{var}-6,\\ \texttt{var}-5\end{array}\right\}, 3\end{array}\right)$$

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.

Graph model We use a similar definition that the one that was utilized for the `minimum` constraint. Within the arc constraint we replace the condition $X < Y$ by the condition $(X \bmod \texttt{M}) < (Y \bmod \texttt{M})$.

See also `minimum`, `maximum_modulo`.

Key words order constraint, modulo, maxint, minimum.

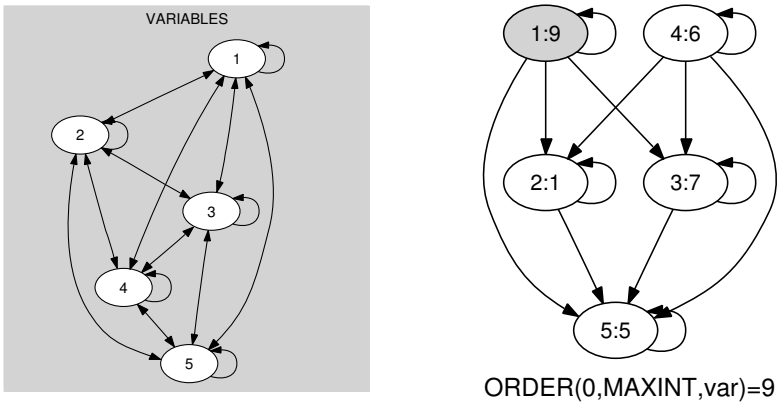


Figure 4.309: Initial and final graph of the `minimum_modulo` constraint

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 ≥ 1
VARIABLES.var ≤ |VARIABLES|
required(MATRIX, [i, j, c])
increasing_seq(MATRIX, [i, j])
MATRIX.i ≥ 1
MATRIX.i ≤ |VARIABLES|
MATRIX.j ≥ 1
MATRIX.j ≤ |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 $CLIQUE \mapsto$ collection(variables1, variables2)

Arc arity 2

Arc constraint(s) variables1.var = variables2.key

Graph property(ies)

- **NTREE** = 0
- **SUM_WEIGHT_ARC**(MATRIX[(variables1.key − 1) ∗ |VARIABLES| + variables1.var].c) = COST

Example

$$\texttt{minimum\_weight\_alldifferent}\left(\left\{\begin{array}{l}\texttt{var}-2,\\ \texttt{var}-3,\\ \texttt{var}-1,\\ \texttt{var}-4\end{array}\right\},\left\{\begin{array}{lll}\texttt{i}-1 & \texttt{j}-1 & \texttt{c}-4,\\ \texttt{i}-1 & \texttt{j}-2 & \texttt{c}-1,\\ \texttt{i}-1 & \texttt{j}-3 & \texttt{c}-7,\\ \texttt{i}-1 & \texttt{j}-4 & \texttt{c}-0,\\ \texttt{i}-2 & \texttt{j}-1 & \texttt{c}-1,\\ \texttt{i}-2 & \texttt{j}-2 & \texttt{c}-0,\\ \texttt{i}-2 & \texttt{j}-3 & \texttt{c}-8,\\ \texttt{i}-2 & \texttt{j}-4 & \texttt{c}-2,\\ \texttt{i}-3 & \texttt{j}-1 & \texttt{c}-3,\\ \texttt{i}-3 & \texttt{j}-2 & \texttt{c}-2,\\ \texttt{i}-3 & \texttt{j}-3 & \texttt{c}-1,\\ \texttt{i}-3 & \texttt{j}-4 & \texttt{c}-6,\\ \texttt{i}-4 & \texttt{j}-1 & \texttt{c}-0,\\ \texttt{i}-4 & \texttt{j}-2 & \texttt{c}-0,\\ \texttt{i}-4 & \texttt{j}-3 & \texttt{c}-6,\\ \texttt{i}-4 & \texttt{j}-4 & \texttt{c}-5\end{array}\right\},17\right)$$

The cost 17 corresponds to the sum $\texttt{MATRIX}[(1-1)\cdot 4+2].\texttt{c}+\texttt{MATRIX}[(2-1)\cdot 4+3].\texttt{c}+\texttt{MATRIX}[(3-1)\cdot 4+1].\texttt{c}+\texttt{MATRIX}[(4-1)\cdot 4+4].\texttt{c} = \texttt{MATRIX}[2].\texttt{c}+\texttt{MATRIX}[7].\texttt{c}+\texttt{MATRIX}[9].\texttt{c}+\texttt{MATRIX}[16].\texttt{c} = 1+8+3+5$. 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.

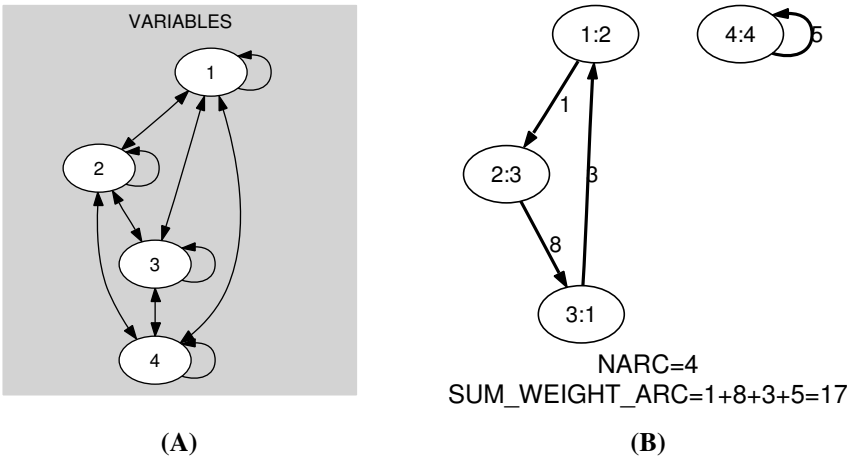


Figure 4.310: Initial and final graph of the minimum_weight_alldifferent constraint

Graph model Since each variable takes one value, and because of the arc constraint $\texttt{variables1} = \texttt{variables.key}$, 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 $\mathbf{NTREE} = 0$, 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 $((i-1) \cdot |\texttt{VARIABLES}| + j)^{th}$ 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 nvalue.

Constraint nclass(NCLASS, VARIABLES, PARTITIONS)

Type(s) VALUES : collection(val − int)

Argument(s)

NCLASS	:	dvar
VARIABLES	:	collection(var − dvar)
PARTITIONS	:	collection(p − VALUES)

Restriction(s)
- required(VALUES, val)
- distinct(VALUES, val)
- NCLASS ≥ 0
- NCLASS $\leq$ min(|VARIABLES|, |PARTITIONS|)
- required(VARIABLES, var)
- required(PARTITIONS, p)
- |PARTITIONS| ≥ 2

Purpose Number of partitions of the collection PARTITIONS such that at least one value is assigned to at least one variable of the collection VARIABLES.

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) **NSCC** = NCLASS

Example

$$\texttt{nclass}\left(\begin{array}{l} 2, \left\{\begin{array}{l}\texttt{var}-3,\\ \texttt{var}-2,\\ \texttt{var}-7,\\ \texttt{var}-2,\\ \texttt{var}-6\end{array}\right\},\\ \left\{\begin{array}{l}\texttt{p}-\{\texttt{val}-1, \texttt{val}-3\},\\ \texttt{p}-\{\texttt{val}-4\},\\ \texttt{p}-\{\texttt{val}-2, \texttt{val}-6\}\end{array}\right\}\end{array}\right)$$

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.

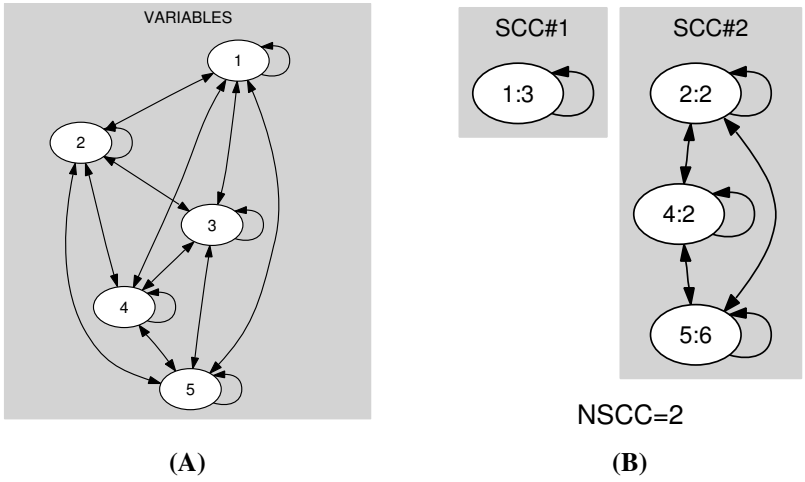


Figure 4.311: Initial and final graph of the `nclass` constraint

4.144 nequivalence

Origin Derived from nvalue.

Constraint nequivalence(NEQUIV, M, VARIABLES)

Argument(s)

NEQUIV	:	dvar
M	:	int
VARIABLES	:	collection(var − dvar)

Restriction(s)

NEQUIV ≥ min(1, |VARIABLES|)
NEQUIV ≤ min(M, |VARIABLES|)
M > 0
required(VARIABLES, var)

Purpose NEQUIV is the number of distinct rests obtained by dividing the variables of the collection VARIABLES by M.

Arc input(s) VARIABLES

Arc generator $CLIQUE \mapsto$ collection(variables1, variables2)

Arc arity 2

Arc constraint(s) variables1.var mod M = variables2.var mod M

Graph property(ies) NSCC = NEQUIV

Example nequivalence$\left(2, 3, \left\{\begin{array}{l} \mathtt{var}-3,\\ \mathtt{var}-2,\\ \mathtt{var}-5,\\ \mathtt{var}-6,\\ \mathtt{var}-15,\\ \mathtt{var}-3,\\ \mathtt{var}-3 \end{array}\right\}\right)$

Parts (A) and (B) of Figure 4.312 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 equivalence class: We have two equivalence classes that respectively correspond to values $\{3, 6, 15\}$ and $\{2, 5\}$.

Algorithm Since constraints $X = Y$ and $X \equiv Y(\bmod M)$ are similar, one should also use a similar algorithm as the one [33, 106] provided for constraint nvalue.

See also nvalue, nclass, ninterval, npair.

Key words counting constraint, value partitioning constraint, number of distinct equivalence classes, strongly connected component, equivalence.

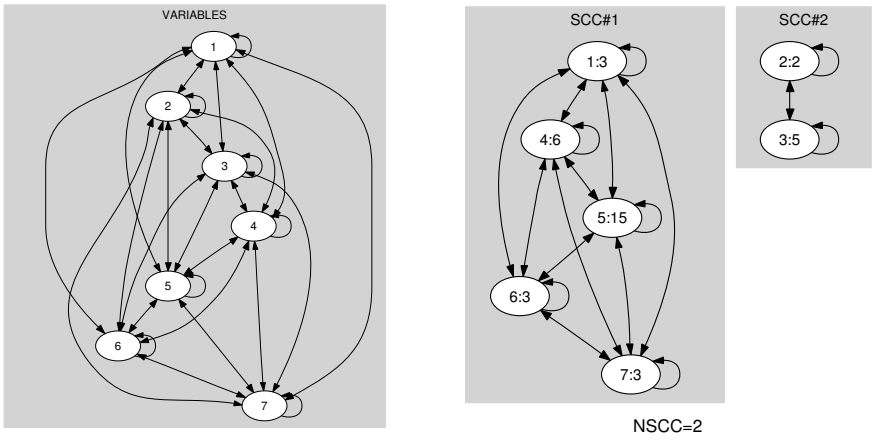


Figure 4.312: Initial and final graph of the `nequivalence` constraint

4.145 next_element

Origin N. Beldiceanu

Constraint next_element(THRESHOLD, INDEX, TABLE, VAL)

Argument(s)

THRESHOLD	:	dvar
INDEX	:	dvar
TABLE	:	collection(index − int, value − dvar)
VAL	:	dvar

Restriction(s)

INDEX ≥ 1
INDEX $\leq |$TABLE$|$
required(TABLE, [index, value])
TABLE.index ≥ 1
TABLE.index $\leq |$TABLE$|$
distinct(TABLE, index)

Purpose INDEX is the smallest entry of TABLE strictly greater than THRESHOLD containing value VAL.

Derived Collection(s)

$$\texttt{col}\left(\begin{array}{l}\texttt{ITEM} - \texttt{collection}(\texttt{index} - \texttt{dvar}, \texttt{value} - \texttt{dvar}),\\ [\texttt{item}(\texttt{index} - \texttt{THRESHOLD}, \texttt{value} - \texttt{VAL})]\end{array}\right)$$

Arc input(s) ITEM TABLE

Arc generator $PRODUCT \mapsto$ collection(item, table)

Arc arity 2

Arc constraint(s)

- item.index $<$ table.index
- item.value $=$ table.value

Graph property(ies) $\mathbf{NARC} > 0$

Sets

$$\begin{array}{l}\texttt{SUCC} \mapsto\\ \left[\begin{array}{l}\texttt{source},\\ \texttt{variables} - \texttt{col}(\texttt{VARIABLES} - \texttt{collection}(\texttt{var} - \texttt{dvar}), [\texttt{item}(\texttt{var} - \texttt{TABLE.index})])\end{array}\right]\end{array}$$

Constraint(s) on sets minimum(INDEX, variables)

Example

$$\texttt{next\_element}\left(\begin{array}{l}2, 3, \left\{\begin{array}{ll}\texttt{index} - 1 & \texttt{value} - 1,\\ \texttt{index} - 2 & \texttt{value} - 8,\\ \texttt{index} - 3 & \texttt{value} - 9,\\ \texttt{index} - 4 & \texttt{value} - 5,\\ \texttt{index} - 5 & \texttt{value} - 9\end{array}\right\}, 9\end{array}\right)$$

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 **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 $\mathtt{I}_k$ and $\mathtt{V}_k$ respectively be the `index` and the `value` attributes of the k^{th} item of the `TABLE` collections. To each quintuple $(\mathtt{THRESHOLD}, \mathtt{INDEX}, \mathtt{VAL}, \mathtt{I}_k, \mathtt{V}_k)$ corresponds a signature variable $\mathtt{S}_k$ as well as the following signature constraint:

$$((\mathtt{I}_k \leq \mathtt{THRESHOLD}) \wedge (\mathtt{I}_k < \mathtt{INDEX}) \wedge (\mathtt{V}_k = \mathtt{VAL})) \Leftrightarrow \mathtt{S}_k = 0 \wedge$$
$$((\mathtt{I}_k \leq \mathtt{THRESHOLD}) \wedge (\mathtt{I}_k < \mathtt{INDEX}) \wedge (\mathtt{V}_k \neq \mathtt{VAL})) \Leftrightarrow \mathtt{S}_k = 1 \wedge$$
$$((\mathtt{I}_k \leq \mathtt{THRESHOLD}) \wedge (\mathtt{I}_k = \mathtt{INDEX}) \wedge (\mathtt{V}_k = \mathtt{VAL})) \Leftrightarrow \mathtt{S}_k = 2 \wedge$$
$$((\mathtt{I}_k \leq \mathtt{THRESHOLD}) \wedge (\mathtt{I}_k = \mathtt{INDEX}) \wedge (\mathtt{V}_k \neq \mathtt{VAL})) \Leftrightarrow \mathtt{S}_k = 3 \wedge$$
$$((\mathtt{I}_k \leq \mathtt{THRESHOLD}) \wedge (\mathtt{I}_k > \mathtt{INDEX}) \wedge (\mathtt{V}_k = \mathtt{VAL})) \Leftrightarrow \mathtt{S}_k = 4 \wedge$$
$$((\mathtt{I}_k \leq \mathtt{THRESHOLD}) \wedge (\mathtt{I}_k > \mathtt{INDEX}) \wedge (\mathtt{V}_k \neq \mathtt{VAL})) \Leftrightarrow \mathtt{S}_k = 5 \wedge$$
$$((\mathtt{I}_k > \mathtt{THRESHOLD}) \wedge (\mathtt{I}_k < \mathtt{INDEX}) \wedge (\mathtt{V}_k = \mathtt{VAL})) \Leftrightarrow \mathtt{S}_k = 6 \wedge$$
$$((\mathtt{I}_k > \mathtt{THRESHOLD}) \wedge (\mathtt{I}_k < \mathtt{INDEX}) \wedge (\mathtt{V}_k \neq \mathtt{VAL})) \Leftrightarrow \mathtt{S}_k = 7 \wedge$$
$$((\mathtt{I}_k > \mathtt{THRESHOLD}) \wedge (\mathtt{I}_k = \mathtt{INDEX}) \wedge (\mathtt{V}_k = \mathtt{VAL})) \Leftrightarrow \mathtt{S}_k = 8 \wedge$$
$$((\mathtt{I}_k > \mathtt{THRESHOLD}) \wedge (\mathtt{I}_k = \mathtt{INDEX}) \wedge (\mathtt{V}_k \neq \mathtt{VAL})) \Leftrightarrow \mathtt{S}_k = 9 \wedge$$
$$((\mathtt{I}_k > \mathtt{THRESHOLD}) \wedge (\mathtt{I}_k > \mathtt{INDEX}) \wedge (\mathtt{V}_k = \mathtt{VAL})) \Leftrightarrow \mathtt{S}_k = 10 \wedge$$
$$((\mathtt{I}_k > \mathtt{THRESHOLD}) \wedge (\mathtt{I}_k > \mathtt{INDEX}) \wedge (\mathtt{V}_k \neq \mathtt{VAL})) \Leftrightarrow \mathtt{S}_k = 11.$$

The automaton is constructed in order to fullfit the following conditions:

- We look for an item of the `TABLE` collection such that $\mathtt{INDEX}_i > \mathtt{THRESHOLD}$ and $\mathtt{INDEX}_i = \mathtt{INDEX}$ and $\mathtt{VALUE}_i = \mathtt{VAL}$,
- There should not exist any item of the `TABLE` collection such that $\mathtt{INDEX}_i > \mathtt{THRESHOLD}$ and $\mathtt{INDEX}_i < \mathtt{INDEX}$ and $\mathtt{VALUE}_i = \mathtt{VAL}$.

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.

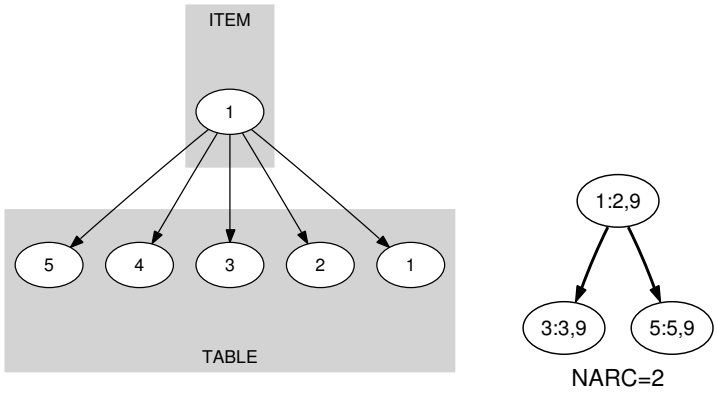


Figure 4.313: Initial and final graph of the `next_element` constraint

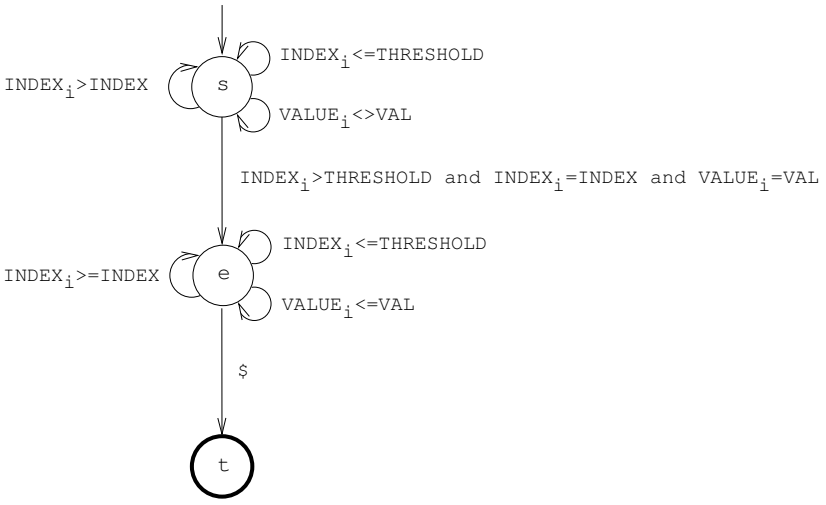


Figure 4.314: Automaton of the `next_element` constraint

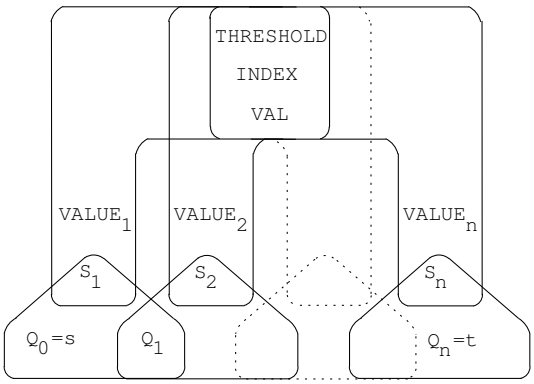


Figure 4.315: Hypergraph of the reformulation corresponding to the automaton of the `next_element` constraint

4.146 next_greater_element

Origin M. Carlsson

Constraint next_greater_element(VAR1, VAR2, VARIABLES)

Argument(s)

VAR1	:	dvar
VAR2	:	dvar
VARIABLES	:	collection(var − dvar)

Restriction(s) |VARIABLES| > 0
required(VARIABLES, var)

Purpose 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.

Derived Collection(s) col(V − collection(var − dvar), [item(var − VAR1)])

Arc input(s) VARIABLES

Arc generator $PATH \mapsto$ collection(variables1, variables2)

Arc arity 2

Arc constraint(s) variables1.var < variables2.var

Graph property(ies) $\mathbf{NARC} = |\text{VARIABLES}| - 1$

Arc input(s) V VARIABLES

Arc generator $PRODUCT \mapsto$ collection(v, variables)

Arc arity 2

Arc constraint(s) v.var < variables.var

Graph property(ies) $\mathbf{NARC} > 0$

Sets SUCC $\mapsto$ [source, variables]

Constraint(s) on sets minimum(VAR2, variables)

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 $\mathbf{NARC}$ graph property, the arcs of the final graph are stressed in bold.

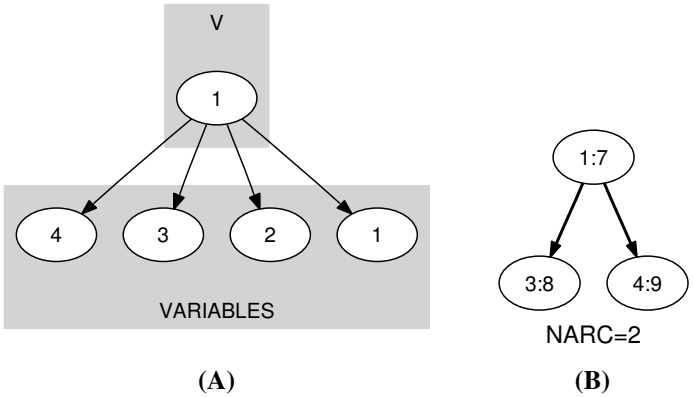


Figure 4.316: Initial and final graph of the next_greater_element constraint

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 |VARIABLES|-1. Therefore the maximum number of arcs of the final graph is equal to |VARIABLES|-1. For this reason we can rewrite $\mathbf{NARC} =$ |VARIABLES| $- 1$ to $\mathbf{NARC} \geq$ |VARIABLES| $- 1$ and simplify $\underline{\mathbf{NARC}}$ to $\overline{\mathbf{NARC}}$.

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 nvalue.

Constraint ninterval(NVAL, VARIABLES, SIZE_INTERVAL)

Argument(s)

NVAL	:	dvar
VARIABLES	:	collection(var − dvar)
SIZE_INTERVAL	:	int

Restriction(s)

NVAL ≥ min(1, |VARIABLES|)
NVAL ≤ |VARIABLES|
required(VARIABLES, var)
SIZE_INTERVAL > 0

Purpose Consider the intervals of the form $[\text{SIZE\_INTERVAL} \cdot k, \text{SIZE\_INTERVAL} \cdot k + \text{SIZE\_INTERVAL} - 1]$ where k is an integer. NVAL is the number of intervals for which at least one value is assigned to at least one variable of the collection VARIABLES.

Arc input(s) VARIABLES

Arc generator $CLIQUE \mapsto$ collection(variables1, variables2)

Arc arity 2

Arc constraint(s) variables1.var/SIZE_INTERVAL = variables2.var/SIZE_INTERVAL

Graph property(ies) **NSCC** = NVAL

Example $$\text{ninterval}\left(\left\langle 2, \left\{\begin{array}{l}\text{var}-3,\\ \text{var}-1,\\ \text{var}-9,\\ \text{var}-1,\\ \text{var}-9\end{array}\right\}, 4\right\rangle\right)$$

Parts (A) and (B) of Figure 4.317 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 those values of an interval which are assigned to some variables of the VARIABLES 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 VARIABLES collection.

Usage The ninterval 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 nvalue, nclass, nequivalence, npair.

Key words counting constraint, value partitioning constraint, number of distinct equivalence classes, interval, strongly connected component, equivalence.

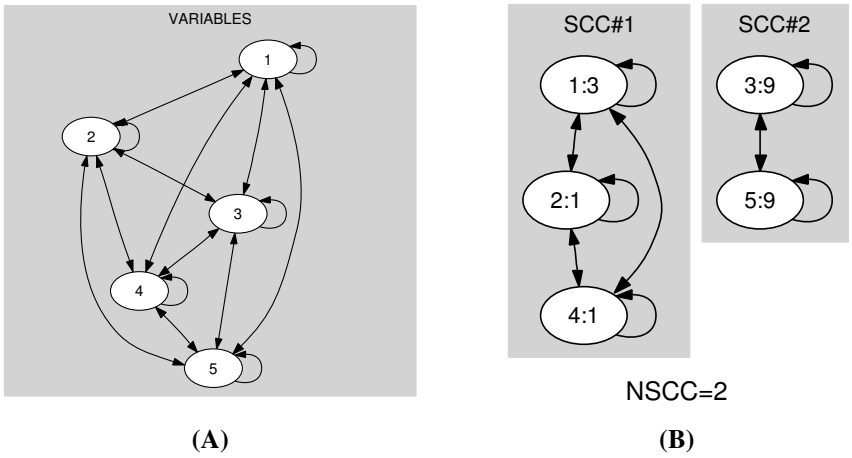


Figure 4.317: Initial and final graph of the `ninterval` constraint

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 $= V_1, \ldots, V_m$ is a *peak* if and only if there exist an i $(1 < i \leq k)$ such that $V_{i-1} < V_i$ and $V_i = V_{i+1} = \ldots = V_k$ and $V_k > V_{k+1}$. The total number of peaks of the sequence of variables VARIABLES is equal to 0.

Example

$$\texttt{no\_peak}\left(\left\langle\begin{array}{l}\texttt{var}-1,\\ \texttt{var}-1,\\ \texttt{var}-4,\\ \texttt{var}-8,\\ \texttt{var}-8\end{array}\right\rangle\right)$$

The previous constraint holds since the sequence 1 1 4 8 8 does not contain any peak.

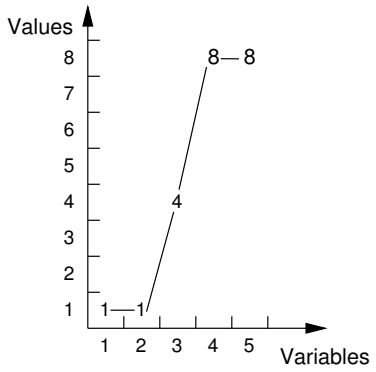


Figure 4.318: A sequence without any peak

Automaton Figure 4.319 depicts the automaton associated to the no_peak constraint. To each pair of consecutive variables $(\texttt{VAR}_i, \texttt{VAR}_{i+1})$ of the collection VARIABLES corresponds a signature variable $\texttt{S}_i$. The following signature constraint links $\texttt{VAR}_i$, $\texttt{VAR}_{i+1}$ and $\texttt{S}_i$: $(\texttt{VAR}_i < \texttt{VAR}_{i+1} \Leftrightarrow \texttt{S}_i = 0) \wedge (\texttt{VAR}_i = \texttt{VAR}_{i+1} \Leftrightarrow \texttt{S}_i = 1) \wedge (\texttt{VAR}_i > \texttt{VAR}_{i+1} \Leftrightarrow \texttt{S}_i = 2)$.

See also peak, no_valley, valley.

Key words sequence, automaton, automaton without counters, sliding cyclic(1) constraint network(1).

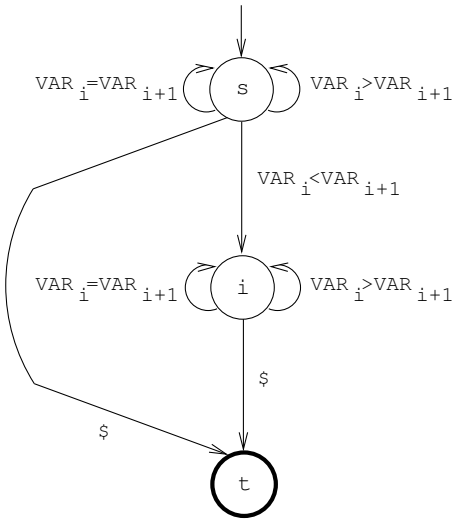


Figure 4.319: Automaton of the no_peak constraint

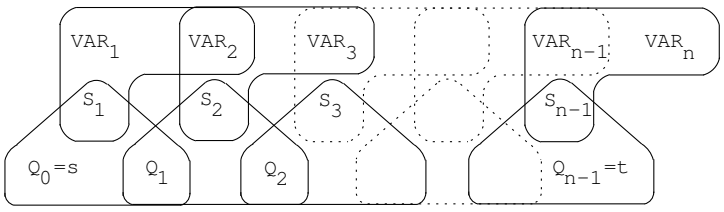


Figure 4.320: Hypergraph of the reformulation corresponding to the automaton of the no_peak constraint

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 $= V_1, \ldots, V_m$ is a *valley* if and only if there exist an i $(1 < i \leq k)$ such that $V_{i-1} > V_i$ and $V_i = V_{i+1} = \ldots = V_k$ and $V_k < V_{k+1}$. The total number of valleys of the sequence of variables VARIABLES is equal to 0.

Example

$$\text{no\_valley}\left(\left\{\begin{array}{l}\text{var}-1,\\ \text{var}-1,\\ \text{var}-4,\\ \text{var}-8,\\ \text{var}-8,\\ \text{var}-2\end{array}\right\}\right)$$

The previous constraint holds since the sequence 1 1 4 8 8 2 does not contain any valley.

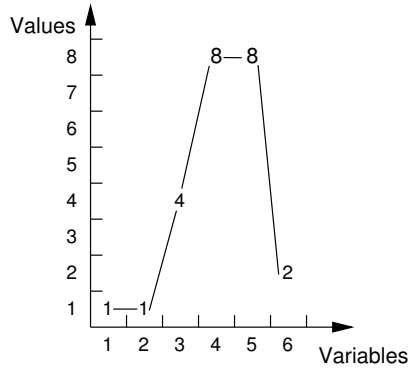


Figure 4.321: A sequence without any valley

Automaton Figure 4.322 depicts the automaton associated to the no_valley constraint. To each pair of consecutive variables $(\text{VAR}_i, \text{VAR}_{i+1})$ of the collection VARIABLES corresponds a signature variable S_i. The following signature constraint links VAR_i, VAR_{i+1} and S_i: $(\text{VAR}_i < \text{VAR}_{i+1} \Leftrightarrow \text{S}_i = 0) \wedge (\text{VAR}_i = \text{VAR}_{i+1} \Leftrightarrow \text{S}_i = 1) \wedge (\text{VAR}_i > \text{VAR}_{i+1} \Leftrightarrow \text{S}_i = 2)$.

See also valley, no_peak, peak.

Key words sequence, automaton, automaton without counters, sliding cyclic(1) constraint network(1).

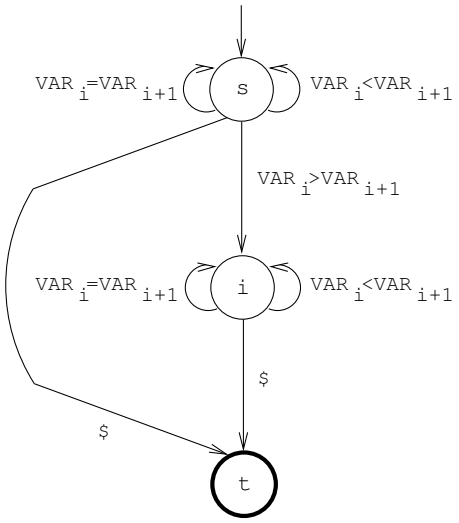


Figure 4.322: Automaton of the `no_valley` constraint

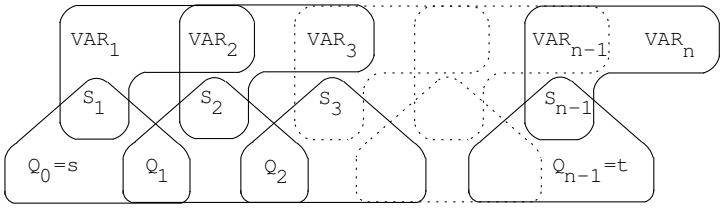


Figure 4.323: Hypergraph of the reformulation corresponding to the automaton of the `no_valley` constraint

4.150 not_all_equal

Origin CHIP

Constraint not_all_equal(VARIABLES)

Argument(s) VARIABLES : collection(var − dvar)

Restriction(s) required(VARIABLES, var)
|VARIABLES| > 1

Purpose The variables of the collection VARIABLES should take more than one single value.

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** > 1

Example not_all_equal $\left(\left\{ \begin{array}{l} \mathtt{var} - 3, \\ \mathtt{var} - 1, \\ \mathtt{var} - 3, \\ \mathtt{var} - 3, \\ \mathtt{var} - 3 \end{array} \right\} \right)$

Parts (A) and (B) of Figure 4.324 respectively show the initial and final graph. Since we use the **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 not_all_equal holds since the final graph contains more than one strongly connected component.

Automaton Figure 4.325 depicts the automaton associated to the not_all_equal constraint. To each pair of consecutive variables $(\mathtt{VAR}_i, \mathtt{VAR}_{i+1})$ of the collection VARIABLES corresponds a signature variable $\mathtt{S}_i$. The following signature constraint links $\mathtt{VAR}_i$, $\mathtt{VAR}_{i+1}$ and $\mathtt{S}_i$: $\mathtt{VAR}_i = \mathtt{VAR}_{i+1} \Leftrightarrow \mathtt{S}_i$.

Algorithm If the intersection of the domains of the variables of the VARIABLES collection is empty the not_all_equal 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 nvalue.

Key words value constraint, disequality, automaton, automaton without counters, sliding cyclic(1) constraint network(1), equivalence.

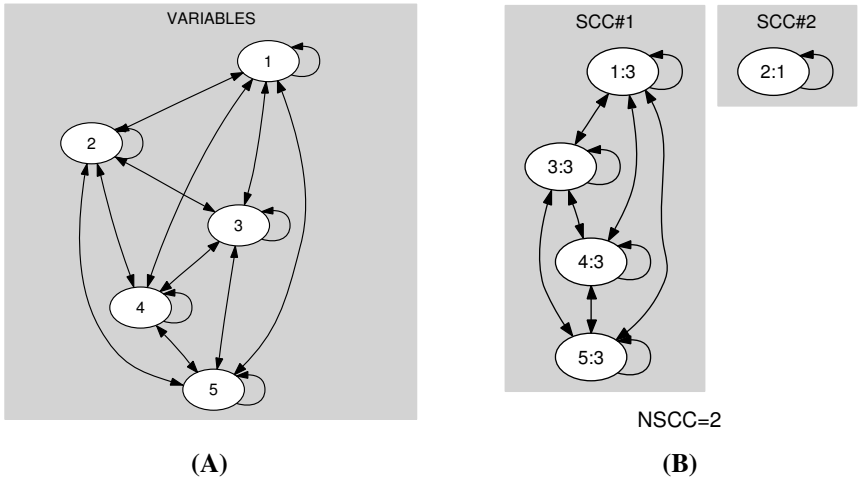


Figure 4.324: Initial and final graph of the `not_all_equal` constraint

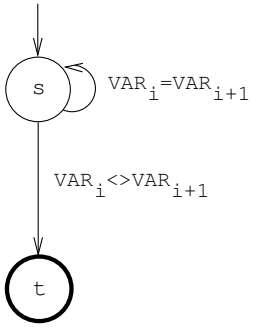


Figure 4.325: Automaton of the `not_all_equal` constraint

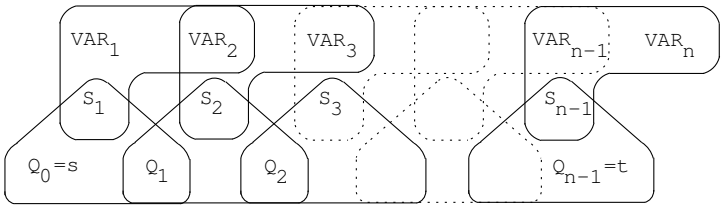


Figure 4.326: Hypergraph of the reformulation corresponding to the automaton of the `not_all_equal` constraint

4.151 not_in

Origin Derived from in.

Constraint not_in(VAR, VALUES)

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) col(VARIABLES − collection(var − dvar), [item(var − VAR)])

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) $\mathbf{NARC} = 0$

Example not_in(2, {val − 1, val − 3})

Figure 4.327 shows the initial graph associated to the previous example. Since we use the $\mathbf{NARC} = 0$ graph property the final graph is empty.

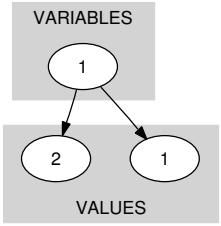


Figure 4.327: Initial graph of the not_in constraint (the final graph is empty)

Signature Since 0 is the smallest number of arcs of the final graph we can rewrite $\mathbf{NARC} = 0$ to $\mathbf{NARC} \leq 0$. This leads to simplify $\overline{\mathbf{NARC}}$ to $\underline{\mathbf{NARC}}$.

Automaton Figure 4.328 depicts the automaton associated to the not_in constraint. Let $\texttt{VAL}_i$ be the val attribute of the i^{th} item of the VALUES collection. To each pair $(\texttt{VAR}, \texttt{VAL}_i)$ corresponds a 0-1 signature variable $\texttt{S}_i$ as well as the following signature constraint: $\texttt{VAR} = \texttt{VAL}_i \Leftrightarrow \texttt{S}_i$.

Remark Entailment occurs immediately after posting this constraint.

Used in group.

See also in.

Key words value constraint, unary constraint, excluded, disequality, domain definition, automaton, automaton without counters, centered cyclic(1) constraint network(1), derived collection.

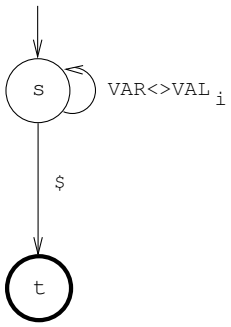


Figure 4.328: Automaton of the not_in constraint

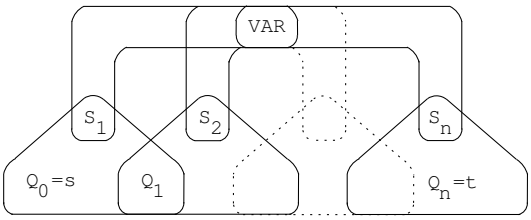


Figure 4.329: Hypergraph of the reformulation corresponding to the automaton of the `not_in` constraint

4.152 npair

Origin Derived from nvalue.

Constraint npair(NVAL, PAIRS)

Argument(s)

NVAL	:	dvar
PAIRS	:	collection(x − dvar, y − dvar)

Restriction(s)

NVAL ≥ min(1, |PAIRS|)
NVAL ≤ |PAIRS|
required(PAIRS, [x, y])

Purpose NVAL is the number of distinct pairs of values assigned to the pairs of variables of the collection PAIRS.

Arc input(s) PAIRS

Arc generator $CLIQUE \mapsto$ collection(pairs1, pairs2)

Arc arity 2

Arc constraint(s)

- pairs1.x = pairs2.x
- pairs1.y = pairs2.y

Graph property(ies) **NSCC** = NVAL

Example

$$\texttt{npair}\left(\begin{array}{ll} 2, \left\{\begin{array}{ll} \texttt{x}-3 & \texttt{y}-1, \\ \texttt{x}-1 & \texttt{y}-5, \\ \texttt{x}-3 & \texttt{y}-1, \\ \texttt{x}-3 & \texttt{y}-1, \\ \texttt{x}-1 & \texttt{y}-5 \end{array}\right\} \end{array}\right)$$

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 PAIRS collection. In our example we have the following pairs of values: (3,1) and (1,5).

Remark This is an example of a *number of distinct values* constraint where there is more than one attribute that is associated to each vertex of the final graph.

See also nvalue, nclass, nequivalence, ninterval.

Key words counting constraint, value partitioning constraint, number of distinct equivalence classes, pair, strongly connected component, equivalence.

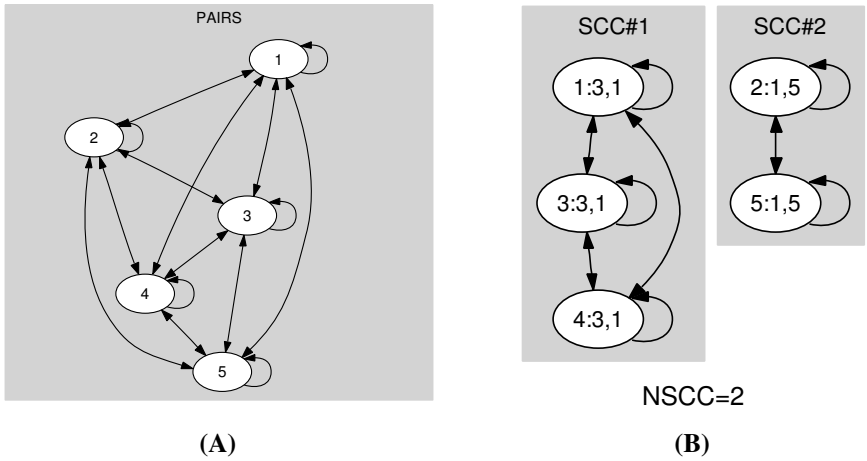


Figure 4.330: Initial and final graph of the `npair` constraint

4.153 nset_of_consecutive_values

Origin N. Beldiceanu

Constraint nset_of_consecutive_values(N, VARIABLES)

Argument(s)

N	:	dvar
VARIABLES	:	collection(var − dvar)

Restriction(s) $\mathtt{N} \geq 1$
$\mathtt{N} \leq |\mathtt{VARIABLES}|$
required(VARIABLES, var)

Purpose N is the number of set of consecutive values used by the variables of the collection VARIABLES.

Arc input(s) VARIABLES

Arc generator $CLIQUE \mapsto$ collection(variables1, variables2)

Arc arity 2

Arc constraint(s) abs(variables1.var − variables2.var) ≤ 1

Graph property(ies) $\mathbf{NSCC} = \mathtt{N}$

Example nset_of_consecutive_values $\left(2, \left\{ \begin{array}{l} \mathtt{var}-3, \\ \mathtt{var}-1, \\ \mathtt{var}-7, \\ \mathtt{var}-1, \\ \mathtt{var}-1, \\ \mathtt{var}-2, \\ \mathtt{var}-8 \end{array} \right\} \right)$

In this example, the variables of the collection VARIABLES 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 **NSCC** graph property, we show the two strongly connected components 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.

Usage Used for specifying the fact that the values have to be used in a compact way is achieved by setting N to 1.

See also min_size_set_of_consecutive_var.

Key words value constraint, consecutive values, strongly connected component.

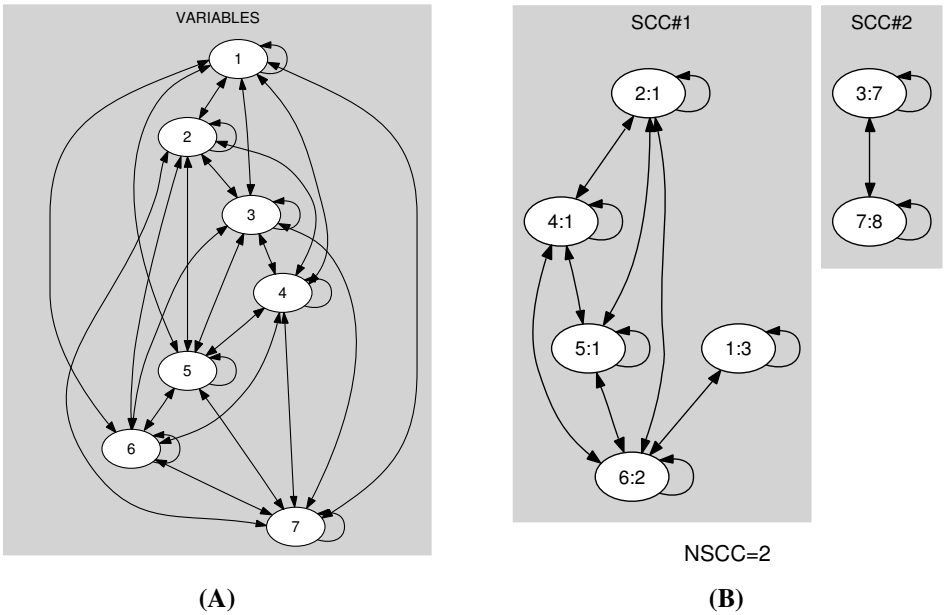


Figure 4.331: Initial and final graph of the `nset_of_consecutive_values` constraint

4.154 nvalue

Origin [73]

Constraint nvalue(NVAL, VARIABLES)

Synonym(s) cardinality_on_attributes_values.

Argument(s)

NVAL	:	dvar
VARIABLES	:	collection(var − dvar)

Restriction(s)

NVAL ≥ min(1, |VARIABLES|)
NVAL ≤ |VARIABLES|
required(VARIABLES, var)

Purpose NVAL is the number of distinct values taken by the variables of the collection VARIABLES.

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

Example

$$\texttt{nvalue}\left(4, \left\{\begin{array}{l}\texttt{var}-3,\\ \texttt{var}-1,\\ \texttt{var}-7,\\ \texttt{var}-1,\\ \texttt{var}-6\end{array}\right\}\right)$$

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.

Automaton Figure 4.333 depicts the automaton associated to the nvalue constraint. To each item of the collection VARIABLES corresponds a signature variable $\texttt{S}_i$, 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 nvalue constraint generalizes several constraints like:

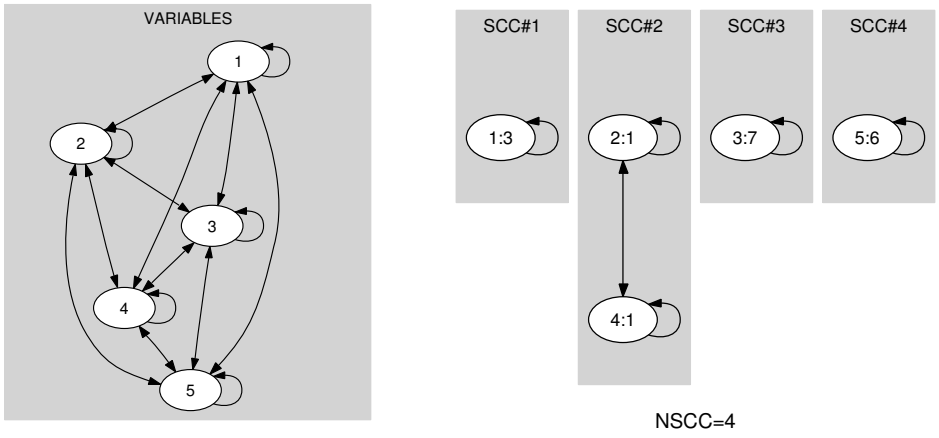


Figure 4.332: Initial and final graph of the `nvalue` constraint

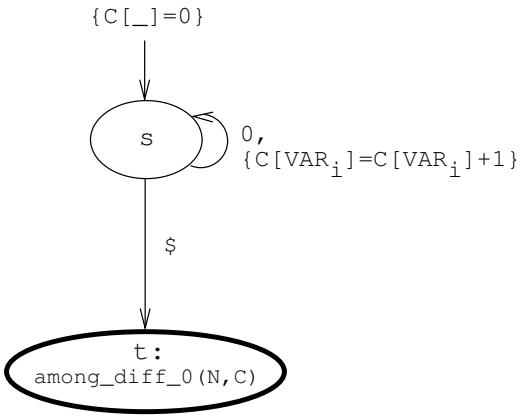


Figure 4.333: Automaton of the `nvalue` constraint

- 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 *Cardinality on Attributes Values*. A constraint called k − diff 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 track.

See also alldifferent, not_all_equal, nvalues, nvalues_except_0, npair, nvalue_on_intersection, among_diff_0.

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.

4.155 nvalue_on_intersection

Origin Derived from common and nvalue.

Constraint nvalue_on_intersection(NVAL, VARIABLES1, VARIABLES2)

Argument(s)

NVAL	:	dvar
VARIABLES1	:	collection(var − dvar)
VARIABLES2	:	collection(var − dvar)

Restriction(s)

NVAL ≥ 0
NVAL ≤ |VARIABLES1|
NVAL ≤ |VARIABLES2|
required(VARIABLES1, var)
required(VARIABLES2, var)

Purpose NVAL is the number of distinct values which both occur in the VARIABLES1 and VARIABLES2 collections.

Arc input(s) VARIABLES1 VARIABLES2

Arc generator $PRODUCT \mapsto$ collection(variables1, variables2)

Arc arity 2

Arc constraint(s) variables1.var = variables2.var

Graph property(ies) **NCC** = NVAL

Example

$$\texttt{nvalue\_on\_intersection}\left(\left\{\begin{array}{l}\texttt{var}-1,\\ \texttt{var}-9,\\ \texttt{var}-1,\\ \texttt{var}-5\end{array}\right\},\left\{\begin{array}{l}\texttt{var}-2,\\ \texttt{var}-1,\\ \texttt{var}-9,\\ \texttt{var}-9,\\ \texttt{var}-6,\\ \texttt{var}-9\end{array}\right\}\right)$$

Parts (A) and (B) of Figure 4.334 respectively show the initial and final graph. Since we use the **NCC** graph property we show the connected components of the final graph. The variable NVAL 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.

See also nvalue, common, alldifferent_on_intersection, same_intersection.

Key words counting constraint, number of distinct values, connected component, constraint on the intersection.

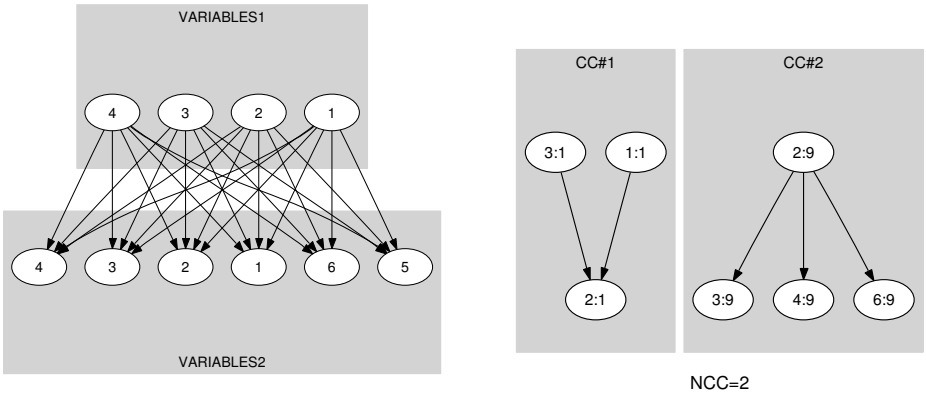


Figure 4.334: Initial and final graph of the `nvalue_on_intersection` constraint

4.156 nvalues

Origin Inspired by nvalue and count.

Constraint nvalues(VARIABLES, RELOP, LIMIT)

Argument(s)

VARIABLES	:	collection(var − dvar)
RELOP	:	atom
LIMIT	:	dvar

Restriction(s) required(VARIABLES, var)
RELOP $\in [=, \neq, <, \geq, >, \leq]$

Purpose Let N be the number of distinct values assigned to the variables of the VARIABLES collection. Enforce condition N RELOP LIMIT to hold.

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** RELOP LIMIT

Example

$$\texttt{nvalues}\left(\left\{\begin{array}{l}\texttt{var}-4,\\ \texttt{var}-5,\\ \texttt{var}-5,\\ \texttt{var}-4,\\ \texttt{var}-1,\\ \texttt{var}-5\end{array}\right\}, =, 3\right)$$

Parts (A) and (B) of Figure 4.335 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 3 following values 1, 4 and 5 are used by the variables of the VARIABLES collection.

Usage Used in the **Constraint(s) on sets** slot for defining some constraints like assign_and_nvalues, circuit_cluster or coloured_cumulative.

Used in assign_and_nvalues, circuit_cluster, coloured_cumulative, coloured_cumulatives.

See also nvalues_except_0, nvalue.

Key words counting constraint, value partitioning constraint, number of distinct equivalence classes, number of distinct values, strongly connected component, equivalence.

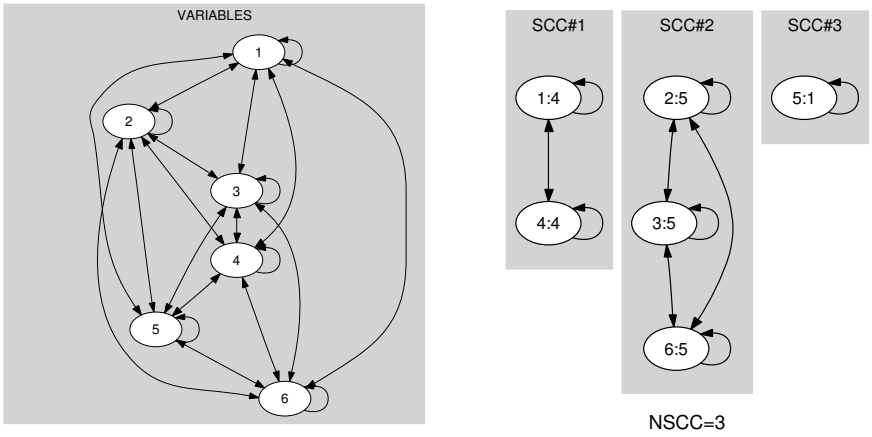


Figure 4.335: Initial and final graph of the `nvalues` constraint

4.157 nvalues_except_0

Origin Derived from nvalues.

Constraint nvalues_except_0(VARIABLES, RELOP, LIMIT)

Argument(s)

VARIABLES	:	collection(var − dvar)
RELOP	:	atom
LIMIT	:	dvar

Restriction(s) required(VARIABLES, var)
RELOP $\in [=, \neq, <, \geq, >, \leq]$

Purpose Let N be the number of distinct values, different from 0, assigned to the variables of the VARIABLES collection. Enforce condition N RELOP LIMIT to hold.

Arc input(s) VARIABLES

Arc generator $CLIQUE \mapsto$ collection(variables1, variables2)

Arc arity 2

Arc constraint(s)
- variables1.var $\neq 0$
- variables1.var = variables2.var

Graph property(ies) **NSCC** RELOP LIMIT

Example

$$\text{nvalues\_except\_0}\left(\left\{\begin{array}{l}\text{var}-4,\\ \text{var}-5,\\ \text{var}-5,\\ \text{var}-4,\\ \text{var}-0,\\ \text{var}-1\end{array}\right\}, =, 3\right)$$

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 VARIABLES collection. Beside value 0, the 3 following values 1, 4 and 5 are assigned to the variables of the VARIABLES collection.

Used in cycle_or_accessibility.

See also nvalues, nvalue, assign_and_nvalues.

Key words counting constraint, value partitioning constraint, number of distinct values, strongly connected component, joker value.

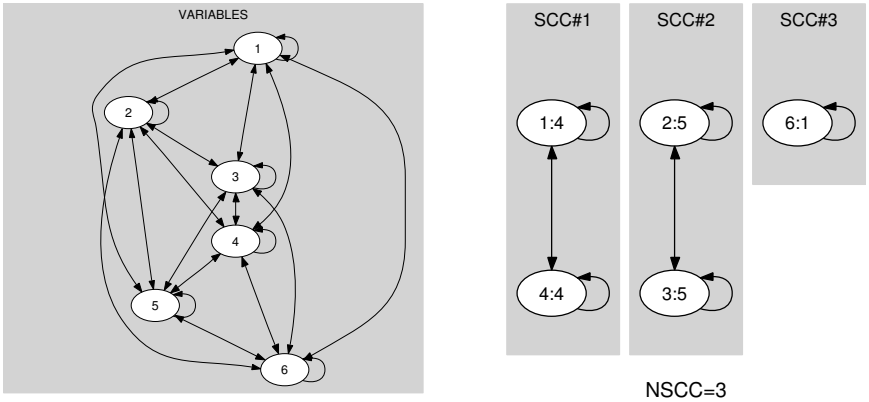


Figure 4.336: Initial and final graph of the `nvalues_except_0` constraint

4.158 one_tree

Origin Inspired by [134]

Constraint one_tree(NODES)

Argument(s)

$$\texttt{NODES} : \texttt{collection}\left(\begin{array}{l}\texttt{id} - \texttt{atom},\\ \texttt{index} - \texttt{int},\\ \texttt{type} - \texttt{int},\\ \texttt{father} - \texttt{dvar},\\ \texttt{depth1} - \texttt{dvar},\\ \texttt{depth2} - \texttt{dvar}\end{array}\right)$$

Restriction(s)

required(NODES, [id, index, type, father, depth1, depth2])
NODES.index ≥ 1
NODES.index $\leq |\texttt{NODES}|$
distinct(NODES, index)
in_list(NODES, type, [2, 3, 6])
NODES.father ≥ 1
NODES.father $\leq |\texttt{NODES}|$
NODES.depth1 ≥ 0
NODES.depth1 $\leq |\texttt{NODES}|$
NODES.depth2 ≥ 0
NODES.depth2 $\leq |\texttt{NODES}|$

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.

Arc input(s) NODES

Arc generator $\mathit{CLIQUE} \mapsto \texttt{collection}(\texttt{nodes1}, \texttt{nodes2})$

Arc arity 2

Arc constraint(s)

$$\bigvee\left(\begin{array}{l}\texttt{nodes1.index} = \texttt{nodes2.index} \wedge \texttt{nodes1.father} = \texttt{nodes1.index},\\ \bigwedge\left(\begin{array}{l}\texttt{nodes1.index} \neq \texttt{nodes2.index},\\ \texttt{nodes1.father} = \texttt{nodes2.index},\\ \bigvee\left(\begin{array}{l}\texttt{nodes1.type} \bmod 2 = 0 \wedge \texttt{nodes1.depth1} > \texttt{nodes2.depth1},\\ \texttt{nodes1.type} \bmod 2 > 0 \wedge \texttt{nodes1.depth1} = \texttt{nodes2.depth1}\end{array}\right),\\ \bigvee\left(\begin{array}{l}\texttt{nodes1.type} \bmod 3 = 0 \wedge \texttt{nodes1.depth2} > \texttt{nodes2.depth2},\\ \texttt{nodes1.type} \bmod 3 > 0 \wedge \texttt{nodes1.depth2} = \texttt{nodes2.depth2}\end{array}\right)\end{array}\right)\end{array}\right)$$

Graph property(ies)

- $\mathbf{MAX\_NSCC} \leq 1$
- $\mathbf{NCC} = 1$
- $\mathbf{NVERTEX} = |\texttt{NODES}|$

Example

$$\texttt{one_tree}\left(\left\{\begin{array}{llllll}
\texttt{id}-\texttt{x} & \texttt{index}-1 & \texttt{type}-2 & \texttt{father}-6 & \texttt{depth1}-2 & \texttt{depth2}-2,\\
\texttt{id}-\texttt{x} & \texttt{index}-2 & \texttt{type}-2 & \texttt{father}-2 & \texttt{depth1}-1 & \texttt{depth2}-0,\\
\texttt{id}-\texttt{x} & \texttt{index}-3 & \texttt{type}-3 & \texttt{father}-6 & \texttt{depth1}-1 & \texttt{depth2}-3,\\
\texttt{id}-\texttt{x} & \texttt{index}-4 & \texttt{type}-3 & \texttt{father}-5 & \texttt{depth1}-2 & \texttt{depth2}-4,\\
\texttt{id}-\texttt{x} & \texttt{index}-5 & \texttt{type}-3 & \texttt{father}-1 & \texttt{depth1}-2 & \texttt{depth2}-3,\\
\texttt{id}-\texttt{x} & \texttt{index}-6 & \texttt{type}-3 & \texttt{father}-7 & \texttt{depth1}-1 & \texttt{depth2}-2,\\
\texttt{id}-\texttt{x} & \texttt{index}-7 & \texttt{type}-3 & \texttt{father}-2 & \texttt{depth1}-1 & \texttt{depth2}-1,\\
\texttt{id}-\texttt{g} & \texttt{index}-8 & \texttt{type}-2 & \texttt{father}-1 & \texttt{depth1}-3 & \texttt{depth2}-2,\\
\texttt{id}-\texttt{a} & \texttt{index}-9 & \texttt{type}-6 & \texttt{father}-4 & \texttt{depth1}-3 & \texttt{depth2}-5,\\
\texttt{id}-\texttt{f} & \texttt{index}-10 & \texttt{type}-6 & \texttt{father}-7 & \texttt{depth1}-2 & \texttt{depth2}-2,\\
\texttt{id}-\texttt{b} & \texttt{index}-11 & \texttt{type}-3 & \texttt{father}-4 & \texttt{depth1}-2 & \texttt{depth2}-5,\\
\texttt{id}-\texttt{c} & \texttt{index}-12 & \texttt{type}-3 & \texttt{father}-5 & \texttt{depth1}-2 & \texttt{depth2}-4,\\
\texttt{id}-\texttt{e} & \texttt{index}-13 & \texttt{type}-3 & \texttt{father}-3 & \texttt{depth1}-1 & \texttt{depth2}-4,\\
\texttt{id}-\texttt{d} & \texttt{index}-14 & \texttt{type}-3 & \texttt{father}-3 & \texttt{depth1}-1 & \texttt{depth2}-4
\end{array}\right\}\right)$$

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`.

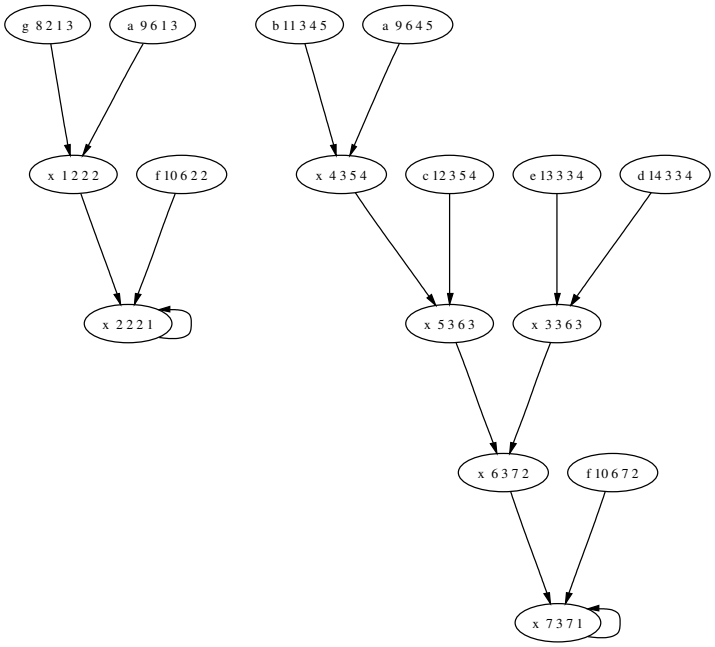


Figure 4.337: The two trees to merge

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:

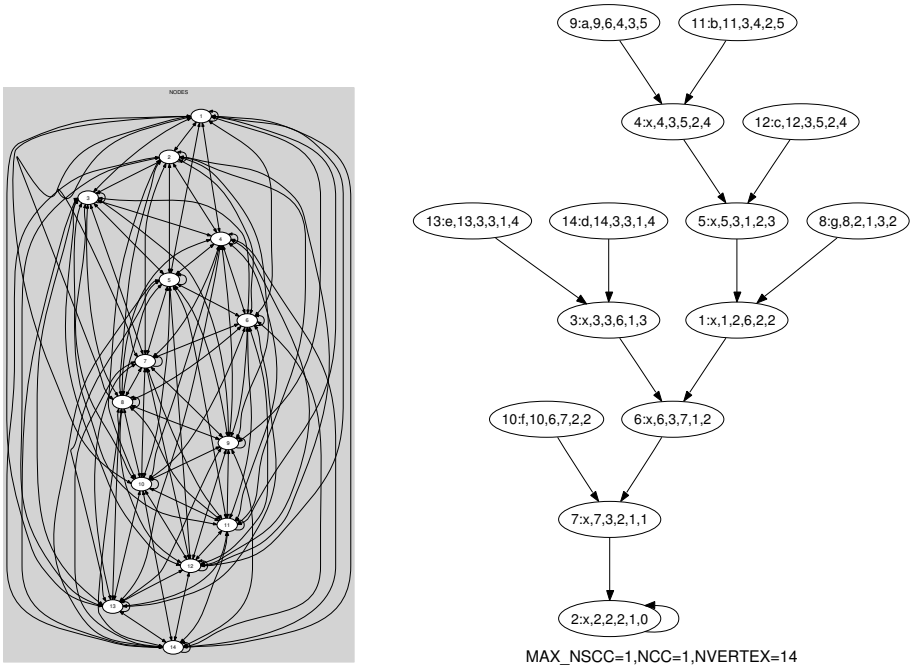


Figure 4.338: Initial and final graph of the `one_tree` constraint

- 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` $\bmod 2 = 0$ (respectively `type` $\bmod 3 = 0$) 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 $\mathbf{MAX\_NSCC} \leq 1$ 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 $\mathbf{NCC} = 1$ imposes having only one single tree.
- Finally the third graph property $\mathbf{NVERTEX} = |\texttt{NODES}|$ 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 $\leq |\text{TREES}|$
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 $CLIQUE(<) \mapsto$ collection(trees1, trees2, trees3)

Arc arity 3

Arc constraint(s)

$$\sum \left(\begin{array}{l} \text{trees1.x} * \text{trees2.y} - \text{trees1.x} * \text{trees3.y}, \\ \text{trees1.y} * \text{trees3.x} - \text{trees1.y} * \text{trees2.x}, \\ \text{trees2.x} * \text{trees3.y} - \text{trees2.y} * \text{trees3.x} \end{array} \right) = 0$$

Graph property(ies) $\mathbf{NARC} = \text{NROW}$

Example

$$\text{orchard}\left(10, \left\{ \begin{array}{lll} \text{index} - 1 & \text{x} - 0 & \text{y} - 0, \\ \text{index} - 2 & \text{x} - 4 & \text{y} - 0, \\ \text{index} - 3 & \text{x} - 8 & \text{y} - 0, \\ \text{index} - 4 & \text{x} - 2 & \text{y} - 4, \\ \text{index} - 5 & \text{x} - 4 & \text{y} - 4, \\ \text{index} - 6 & \text{x} - 6 & \text{y} - 4, \\ \text{index} - 7 & \text{x} - 0 & \text{y} - 8, \\ \text{index} - 8 & \text{x} - 4 & \text{y} - 8, \\ \text{index} - 9 & \text{x} - 8 & \text{y} - 8 \end{array} \right\} \right)$$

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 $(\texttt{trees}_1\texttt{.x}, \texttt{trees}_1\texttt{.y})$, $(\texttt{trees}_2\texttt{.x}, \texttt{trees}_2\texttt{.y})$ and $(\texttt{trees}_3\texttt{.x}, \texttt{trees}_3\texttt{.y})$ are aligned. It corresponds to the development of the expression:

$$\begin{vmatrix} \texttt{trees}_1\texttt{.x} & \texttt{trees}_2\texttt{.y} & 1 \\ \texttt{trees}_2\texttt{.x} & \texttt{trees}_2\texttt{.y} & 1 \\ \texttt{trees}_3\texttt{.x} & \texttt{trees}_3\texttt{.y} & 1 \end{vmatrix} = 0$$

Key words geometrical constraint, alignment, hypergraph.

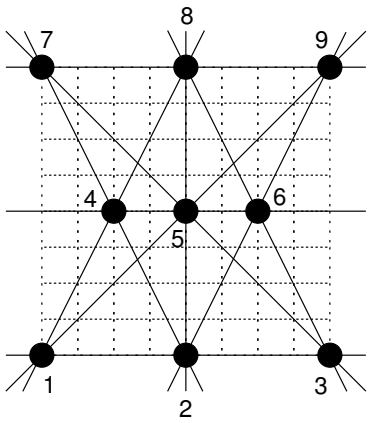


Figure 4.339: Nine trees with 10 alignments of 3 trees

4.160 orth_link_ori_siz_end

Origin Used by several constraints between orthotopes

Constraint orth_link_ori_siz_end(ORTHOTOPE)

Argument(s) ORTHOTOPE : collection(ori − dvar, siz − dvar, end − dvar)

Restriction(s) |ORTHOTOPE| > 0
require_at_least(2, ORTHOTOPE, [ori, siz, end])
ORTHOTOPE.siz ≥ 0

Purpose Enforce for each item of the ORTHOTOPE collection the constraint ori + siz = end.

Arc input(s) ORTHOTOPE

Arc generator $SELF \mapsto$ collection(orthotope)

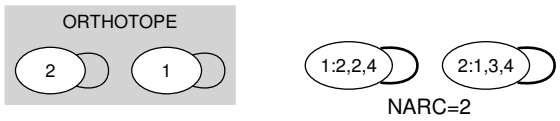
Arc arity 1

Arc constraint(s) orthotope.ori + orthotope.siz = orthotope.end

Graph property(ies) $\mathbf{NARC} = |\texttt{ORTHOTOPE}|$

Example $\texttt{orth\_link\_ori\_siz\_end}\left(\left\{\begin{array}{lll}\texttt{ori}-2 & \texttt{siz}-2 & \texttt{end}-4,\\ \texttt{ori}-1 & \texttt{siz}-3 & \texttt{end}-4\end{array}\right\}\right)$

Parts (A) and (B) of Figure 4.340 respectively show the initial and final graph. Since we use the $\mathbf{NARC}$ graph property, the unary arcs of the final graph are stressed in bold.



(A) **(B)**

Figure 4.340: Initial and final graph of the orth_link_ori_siz_end constraint

Signature Since we use the *SELF* arc generator on the ORTHOTOPE collection the number of arcs of the initial graph is equal to |ORTHOTOPE|. Therefore the maximum number of arcs of the final graph is also equal to |ORTHOTOPE|. For this reason we can rewrite the graph property $\mathbf{NARC} = |\texttt{ORTHOTOPE}|$ to $\mathbf{NARC} \geq |\texttt{ORTHOTOPE}|$ and simplify $\underline{\overline{\mathbf{NARC}}}$ to $\overline{\mathbf{NARC}}$.

Usage Used in the **Arc constraint(s)** slot for defining some constraints like diffn, place_in_pyramid or orths_are_connected.

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_DIMth item of the ORTHOTOPES collection should be fixed to one.

Arc input(s) ORTHOTOPE

Arc generator $SELF \mapsto$ collection(orthotope)

Arc arity 1

Arc constraint(s)
- orthotope.key = VERTICAL_DIM
- orthotope.ori = 1

Graph property(ies) **NARC** = 1

Example $\texttt{orth\_on\_the\_ground}\left(\left\{\begin{array}{lll}\texttt{ori}-1 & \texttt{siz}-2 & \texttt{end}-3,\\ \texttt{ori}-2 & \texttt{siz}-3 & \texttt{end}-5\end{array}\right\}, 1\right)$

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.

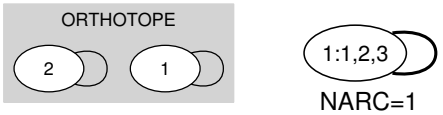


Figure 4.341: Initial and final graph of the orth_on_the_ground constraint

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 $SELF$ arc generator the final graph contains at most one arc. Therefore we can rewrite the graph property $\mathbf{NARC} = 1$ to $\mathbf{NARC} \geq 1$ and simplify $\underline{\overline{\mathbf{NARC}}}$ to $\overline{\mathbf{NARC}}$.

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 place_in_pyramid.

Constraint orth_on_top_of_orth(ORTHOTOPE1, ORTHOTOPE2, VERTICAL_DIM)

Type(s) ORTHOTOPE : collection(ori − dvar, siz − dvar, end − dvar)

Argument(s)

ORTHOTOPE1	:	ORTHOTOPE
ORTHOTOPE2	:	ORTHOTOPE
VERTICAL_DIM	:	int

Restriction(s)
- |ORTHOTOPE| > 0
- require_at_least(2, ORTHOTOPE, [ori, siz, end])
- ORTHOTOPE.siz ≥ 0
- |ORTHOTOPE1| = |ORTHOTOPE2|
- VERTICAL_DIM ≥ 1
- VERTICAL_DIM ≤ |ORTHOTOPE1|
- orth_link_ori_siz_end(ORTHOTOPE1)
- orth_link_ori_siz_end(ORTHOTOPE2)

Purpose

ORTHOTOPE1 is located on top of ORTHOTOPE2 which concretely means:
- In each dimension different from VERTICAL_DIM the projection of ORTHOTOPE1 is included in the projection of ORTHOTOPE2.
- In the dimension VERTICAL_DIM the origin of ORTHOTOPE1 coincide with the end of ORTHOTOPE2.

Arc input(s) ORTHOTOPE1 ORTHOTOPE2

Arc generator $PRODUCT(=) \mapsto$ collection(orthotope1, orthotope2)

Arc arity 2

Arc constraint(s)
- orthotope1.key ≠ VERTICAL_DIM
- orthotope2.ori ≤ orthotope1.ori
- orthotope1.end ≤ orthotope2.end

Graph property(ies) NARC = |ORTHOTOPE1| − 1

Arc input(s) ORTHOTOPE1 ORTHOTOPE2

Arc generator $PRODUCT(=) \mapsto$ collection(orthotope1, orthotope2)

Arc arity 2

Arc constraint(s)
- orthotope1.key = VERTICAL_DIM
- orthotope1.ori = orthotope2.end

Graph property(ies) $\mathbf{NARC} = 1$

Example $\texttt{orth\_on\_top\_of\_orth}\left(\left\{\begin{array}{lll}\texttt{ori}-5 & \texttt{siz}-2 & \texttt{end}-7,\\ \texttt{ori}-3 & \texttt{siz}-3 & \texttt{end}-6\end{array}\right\},\left\{\begin{array}{lll}\texttt{ori}-3 & \texttt{siz}-5 & \texttt{end}-8,\\ \texttt{ori}-1 & \texttt{siz}-2 & \texttt{end}-3\end{array}\right\},2\right)$

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 $\mathbf{NARC}$ graph property, the unique arc of the final graph is stressed in bold.

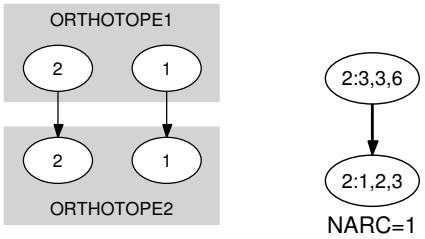


Figure 4.342: Initial and final graph of the `orth_on_top_of_orth` constraint

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 $\mathbf{NARC} = 1$ to $\mathbf{NARC} \geq 1$ and simplify $\underline{\overline{\mathbf{NARC}}}$ to $\overline{\mathbf{NARC}}$.

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 orths_are_connected(ORTHOTOPES)

Type(s) ORTHOTOPE : collection(ori − dvar, siz − dvar, end − dvar)

Argument(s) ORTHOTOPES : collection(orth − ORTHOTOPE)

Restriction(s) |ORTHOTOPE| > 0
require_at_least(2, ORTHOTOPE, [ori, siz, end])
ORTHOTOPE.siz > 0
required(ORTHOTOPES, orth)
same_size(ORTHOTOPES, orth)

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) ORTHOTOPES

Arc generator $SELF \mapsto$ collection(orthotopes)

Arc arity 1

Arc constraint(s) orth_link_ori_siz_end(orthotopes.orth)

Graph property(ies) $\mathbf{NARC} = |$ORTHOTOPES$|$

Arc input(s) ORTHOTOPES

Arc generator $CLIQUE(\neq) \mapsto$ collection(orthotopes1, orthotopes2)

Arc arity 2

Arc constraint(s) two_orth_are_in_contact(orthotopes1.orth, orthotopes2.orth)

Graph property(ies)
- $\mathbf{NVERTEX} = |$ORTHOTOPES$|$
- $\mathbf{NCC} = 1$

Example

$$\texttt{orths\_are\_connected}\left(\left\{\begin{array}{l}\left\{\begin{array}{llll}\texttt{orth} - \{ & \texttt{ori} - 2 & \texttt{siz} - 4 & \texttt{end} - 6,\\ & \texttt{ori} - 2 & \texttt{siz} - 2 & \texttt{end} - 4 \}\end{array}\right\},\\ \left\{\begin{array}{llll}\texttt{orth} - \{ & \texttt{ori} - 1 & \texttt{siz} - 2 & \texttt{end} - 3,\\ & \texttt{ori} - 4 & \texttt{siz} - 3 & \texttt{end} - 7 \}\end{array}\right\},\\ \left\{\begin{array}{llll}\texttt{orth} - \{ & \texttt{ori} - 7 & \texttt{siz} - 4 & \texttt{end} - 11,\\ & \texttt{ori} - 1 & \texttt{siz} - 2 & \texttt{end} - 3 \}\end{array}\right\},\\ \left\{\begin{array}{llll}\texttt{orth} - \{ & \texttt{ori} - 6 & \texttt{siz} - 2 & \texttt{end} - 8,\\ & \texttt{ori} - 3 & \texttt{siz} - 2 & \texttt{end} - 5 \}\end{array}\right\}\end{array}\right\}\right)$$

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.

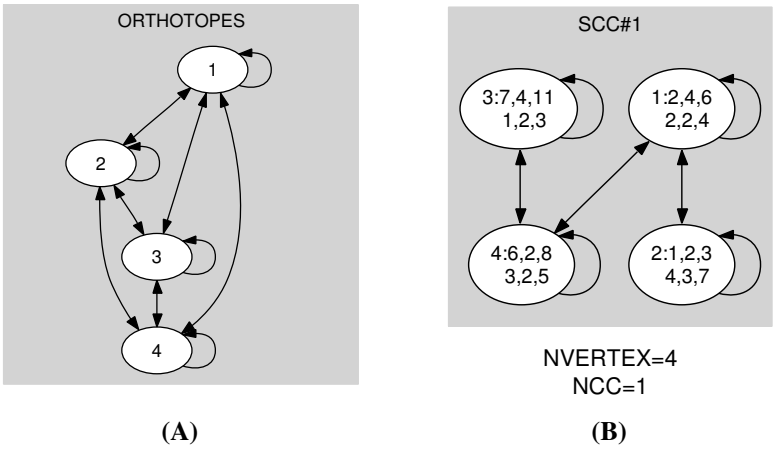


Figure 4.343: Initial and final graph of the `orths_are_connected` constraint

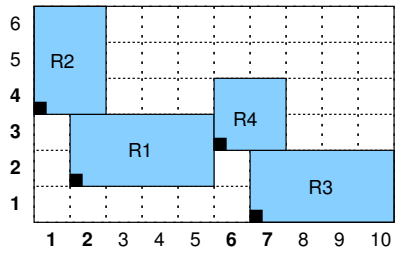


Figure 4.344: Four connected rectangles

Signature

Since the first graph constraint uses the *SELF* arc generator on the `ORTHOTOPES` collection the corresponding initial graph contains |`ORTHOTOPES`| arcs. Therefore the final graph of the first graph constraint contains at most |`ORTHOTOPES`| arcs and we can rewrite $\mathbf{NARC} = |\texttt{ORTHOTOPES}|$ to $\mathbf{NARC} \geq |\texttt{ORTHOTOPES}|$. So we can simplify $\overline{\mathbf{NARC}}$ to $\overline{\mathbf{NARC}}$.

Consider now the second graph constraint. Since its corresponding initial graph contains |ORTHOTOPES| vertices, its final graph has a maximum number of vertices also equal to |ORTHOTOPES|. Therefore we can rewrite $\mathbf{NVERTEX} = |\texttt{ORTHOTOPES}|$ to $\mathbf{NVERTEX} \geq |\texttt{ORTHOTOPES}|$ and simplify $\overline{\underline{\mathbf{NVERTEX}}}$ to $\overline{\mathbf{NVERTEX}}$. From the graph property $\mathbf{NVERTEX} = |\texttt{ORTHOTOPES}|$ and from the restriction $|\texttt{ORTHOTOPES}| > 0$ 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 $\overline{\underline{\mathbf{NCC}}}$ to $\underline{\mathbf{NCC}}$.

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 `two_orth_are_in_contact`.

Key words geometrical constraint, touch, contact, non-overlapping, orthotope.

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 $\leq |$NODES$|$
TO ≥ 1
TO $\leq |$NODES$|$
required(NODES, [index, succ])
NODES.index ≥ 1
NODES.index $\leq |$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) **PATH_FROM_TO**(index, FROM, TO) $= 1$

Example

$$\texttt{path\_from\_to}\left(\begin{array}{l} 4, 3, \left\{\begin{array}{ll} \texttt{index} - 1 & \texttt{succ} - \emptyset, \\ \texttt{index} - 2 & \texttt{succ} - \emptyset, \\ \texttt{index} - 3 & \texttt{succ} - \{5\}, \\ \texttt{index} - 4 & \texttt{succ} - \{5\}, \\ \texttt{index} - 5 & \texttt{succ} - \{2, 3\} \end{array}\right\} \end{array}\right)$$

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 $\mathbf{PATH\_FROM\_TO}(\texttt{index}, \texttt{FROM}, \texttt{TO}) = 1$ to $\underline{\mathbf{PATH\_FROM\_TO}}(\texttt{index}, \texttt{FROM}, \texttt{TO}) \geq 1$. Therefore we simplify $\underline{\mathbf{PATH\_FROM\_TO}}$ to $\mathbf{PATH\_FROM\_TO}$.

See also temporal_path, link_set_to_booleans.

Key words graph constraint, path, linear programming, constraint involving set variables.

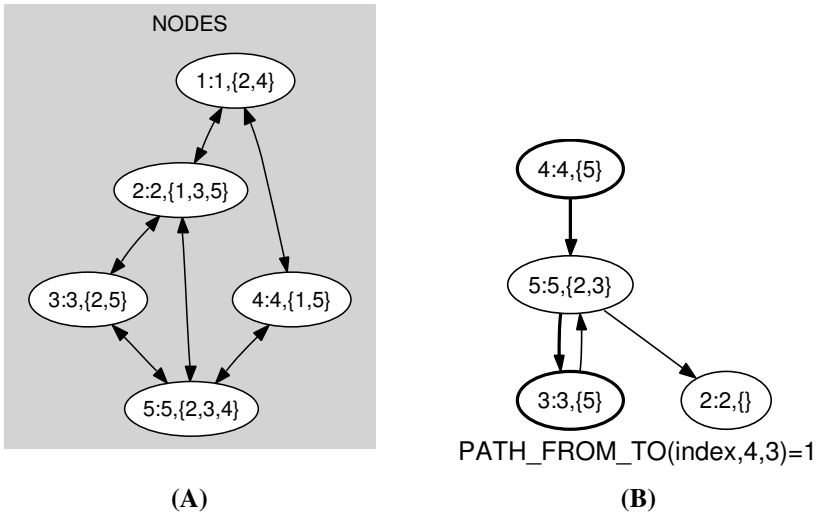


Figure 4.345: Initial and final graph of the `path_from_to` set constraint

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 $V = \{v_1, v_2, \ldots, v_m\}$ and a sequence $\mathbf{s} = \langle s_1, s_2, \ldots, s_n \rangle$ of elements of V. Consider now the sequence $\langle v_{i1}, v_{i2}, \ldots, v_{il} \rangle$ of the types of the successive stretches that appear in $\mathbf{s}$. Let $\mathcal{P}$ be a set of k-pattern. Vector $\mathbf{s}$ satisfies $\mathcal{P}$ if and only if every subsequence of k elements in $\langle v_{i1}, v_{i2}, \ldots, v_{il} \rangle$ belongs to $\mathcal{P}$.

Example

$$\texttt{pattern}\left(\begin{array}{l} \left\{\begin{array}{l} \texttt{var}-1,\\ \texttt{var}-1,\\ \texttt{var}-2,\\ \texttt{var}-2,\\ \texttt{var}-2,\\ \texttt{var}-1,\\ \texttt{var}-3,\\ \texttt{var}-3 \end{array}\right\},\\ \left\{\begin{array}{l} \texttt{pat}-\{\texttt{var}-1, \texttt{var}-2, \texttt{var}-1\},\\ \texttt{pat}-\{\texttt{var}-1, \texttt{var}-2, \texttt{var}-3\},\\ \texttt{pat}-\{\texttt{var}-2, \texttt{var}-1, \texttt{var}-3\} \end{array}\right\} \end{array}\right)$$

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 inflexion.

Constraint peak(N, VARIABLES)

Argument(s)

N	:	dvar
VARIABLES	:	collection(var − dvar)

Restriction(s)

$\texttt{N} \geq 0$
$2 * \texttt{N} \leq \texttt{max}(|\texttt{VARIABLES}| - 1, 0)$
required(VARIABLES, var)

Purpose

A variable V_k $(1 < k < m)$ of the sequence of variables $\texttt{VARIABLES} = V_1, \ldots, V_m$ is a *peak* if and only if there exist an i $(1 < i \leq k)$ such that $V_{i-1} < V_i$ and $V_i = V_{i+1} = \ldots = V_k$ and $V_k > V_{k+1}$. N is the total number of peaks of the sequence of variables VARIABLES.

Example

$$\texttt{peak}\left(2, \left\{\begin{array}{l} \texttt{var} - 1, \\ \texttt{var} - 1, \\ \texttt{var} - 4, \\ \texttt{var} - 8, \\ \texttt{var} - 6, \\ \texttt{var} - 2, \\ \texttt{var} - 7, \\ \texttt{var} - 1 \end{array}\right\}\right)$$

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.

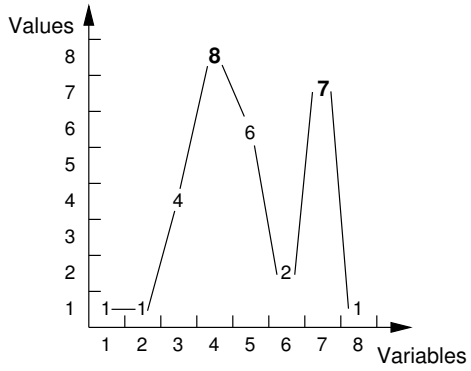


Figure 4.346: The sequence and its two peaks

Automaton Figure 4.347 depicts the automaton associated to the peak constraint. To each pair of consecutive variables $(\texttt{VAR}_i, \texttt{VAR}_{i+1})$ of the collection VARIABLES corresponds a signature variable $\texttt{S}_i$. The following signature constraint links $\texttt{VAR}_i$, $\texttt{VAR}_{i+1}$ and $\texttt{S}_i$: $(\texttt{VAR}_i > \texttt{VAR}_{i+1} \Leftrightarrow \texttt{S}_i = 0) \wedge (\texttt{VAR}_i = \texttt{VAR}_{i+1} \Leftrightarrow \texttt{S}_i = 1) \wedge (\texttt{VAR}_i < \texttt{VAR}_{i+1} \Leftrightarrow \texttt{S}_i = 2)$.

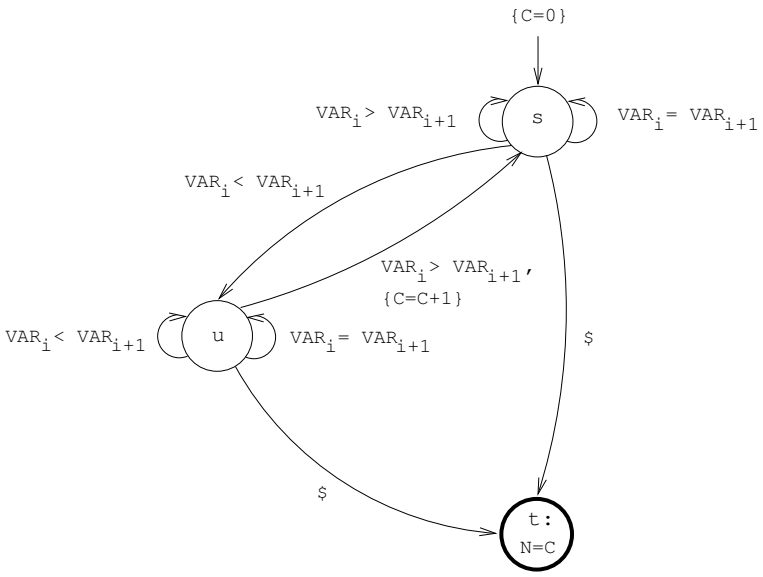


Figure 4.347: Automaton of the peak constraint

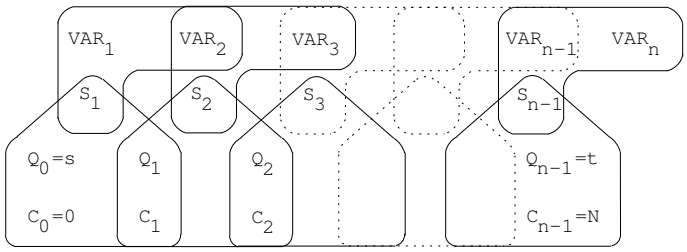


Figure 4.348: Hypergraph of the reformulation corresponding to the automaton of the peak constraint

Usage Useful for constraining the number of *peaks* of a sequence of domain variables.

Remark Since the arity of the arc constraint is not fixed, the `peak` 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 `no_peak`, `inflexion`, `valley`.

Key words sequence, automaton, automaton with counters, sliding cyclic(1) constraint network(2).

4.167 period

Origin N. Beldiceanu

Constraint period(PERIOD, VARIABLES, CTR)

Argument(s)

PERIOD	:	dvar
VARIABLES	:	collection(var − dvar)
CTR	:	atom

Restriction(s)

PERIOD ≥ 1
PERIOD $\leq |$VARIABLES$|$
required(VARIABLES, var)
CTR $\in [=, \neq, <, \geq, >, \leq]$

Purpose

Let us note $V_0, V_1, \ldots, V_{m-1}$ the variables of the VARIABLES collection. PERIOD is the *period* of the sequence $V_0\ V_1 \ldots V_{m-1}$ according to constraint CTR . This means that PERIOD is the smallest natural number such that V_i CTR $V_{i+\texttt{PERIOD}}$ holds for all $i \in 0, 1, \ldots, m - \texttt{PERIOD} - 1$.

Example

$$\texttt{period}\left(\begin{array}{l} 3, \left\{\begin{array}{l} \texttt{var}-1,\\ \texttt{var}-1,\\ \texttt{var}-4,\\ \texttt{var}-1,\\ \texttt{var}-1,\\ \texttt{var}-4,\\ \texttt{var}-1,\\ \texttt{var}-1 \end{array}\right\}, = \end{array}\right)$$

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.

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 $\leq |$VARIABLES$|$
required(VARIABLES, var)
CTR $\in [=, \neq, <, \geq, >, \leq]$

Purpose

Let us note $V_0, V_1, \ldots, V_{m-1}$ the variables of the VARIABLES collection. PERIOD is the *period* of the sequence $V_0\ V_1 \ldots V_{m-1}$ according to constraint CTR . This means that PERIOD is the smallest natural number such that V_i CTR $V_{i+\text{PERIOD}} \vee V_i = 0 \vee V_{i+\text{PERIOD}} = 0$ holds for all $i \in 0, 1, \ldots, m - \text{PERIOD} - 1$.

Example

$$\text{period\_except\_0}\left(\begin{array}{c} 3, \left\{\begin{array}{l} \text{var}-1,\\ \text{var}-1,\\ \text{var}-4,\\ \text{var}-1,\\ \text{var}-1,\\ \text{var}-0,\\ \text{var}-1,\\ \text{var}-1 \end{array}\right\}, = \end{array}\right)$$

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.

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)

$$\bigvee \left(\begin{array}{l} \bigwedge \left(\begin{array}{l} \texttt{orthotopes1.key} = \texttt{orthotopes2.key}, \\ \texttt{orth\_on\_the\_ground}(\texttt{orthotopes1.orth}, \texttt{VERTICAL\_DIM}) \end{array} \right), \\ \bigwedge \left(\begin{array}{l} \texttt{orthotopes1.key} \neq \texttt{orthotopes2.key}, \\ \texttt{orth\_on\_top\_of\_orth}(\texttt{orthotopes1.orth}, \texttt{orthotopes2.orth}, \texttt{VERTICAL\_DIM}) \end{array} \right) \end{array} \right)$$

Graph property(ies) NARC = |ORTHOTOPES|

Example

$$\texttt{place\_in\_pyramid}\left(\left\{ \begin{array}{llll} \texttt{orth} - \{ & \texttt{ori} - 1 & \texttt{siz} - 3 & \texttt{end} - 4, \\ & \texttt{ori} - 1 & \texttt{siz} - 2 & \texttt{end} - 3 \}, \\ \texttt{orth} - \{ & \texttt{ori} - 1 & \texttt{siz} - 2 & \texttt{end} - 3, \\ & \texttt{ori} - 3 & \texttt{siz} - 3 & \texttt{end} - 6 \}, \\ \texttt{orth} - \{ & \texttt{ori} - 5 & \texttt{siz} - 6 & \texttt{end} - 11, \\ & \texttt{ori} - 1 & \texttt{siz} - 2 & \texttt{end} - 3 \}, \\ \texttt{orth} - \{ & \texttt{ori} - 5 & \texttt{siz} - 2 & \texttt{end} - 7, \\ & \texttt{ori} - 3 & \texttt{siz} - 2 & \texttt{end} - 5 \}, \\ \texttt{orth} - \{ & \texttt{ori} - 8 & \texttt{siz} - 3 & \texttt{end} - 11, \\ & \texttt{ori} - 3 & \texttt{siz} - 2 & \texttt{end} - 5 \}, \\ \texttt{orth} - \{ & \texttt{ori} - 8 & \texttt{siz} - 2 & \texttt{end} - 10, \\ & \texttt{ori} - 5 & \texttt{siz} - 2 & \texttt{end} - 7 \} \end{array} \right\}, 2 \right)$$

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.

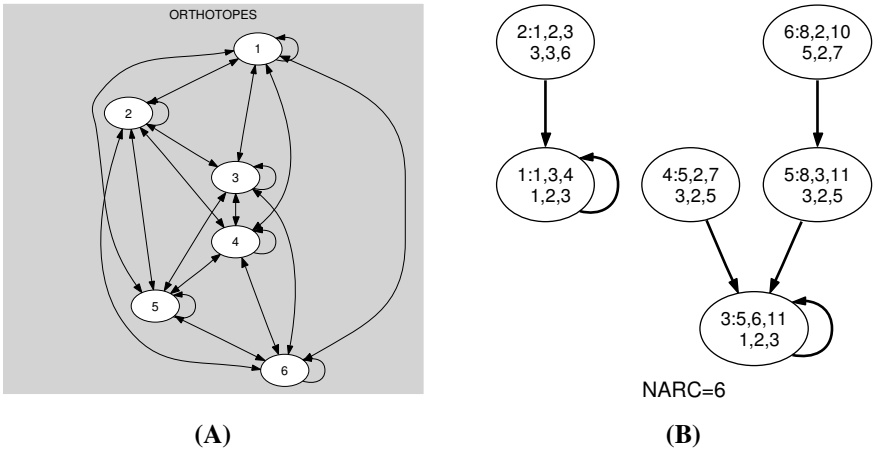


Figure 4.349: Initial and final graph of the `place_in_pyramid` constraint

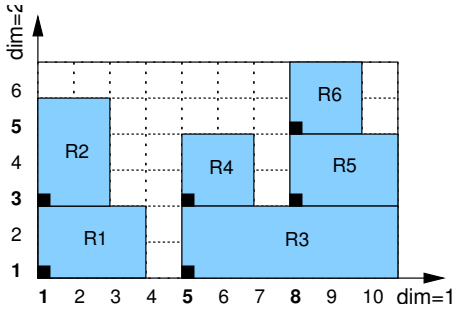


Figure 4.350: Solution corresponding to the final graph

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.

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 $\leq |$CELLS$|$
$|$CELLS$| \geq 1$
required(CELLS, [index, right, left, up, down])
distinct(CELLS, index)
CELLS.right ≥ 0
CELLS.right $\leq |$CELLS$|$
CELLS.left ≥ 0
CELLS.left $\leq |$CELLS$|$
CELLS.up ≥ 0
CELLS.up $\leq |$CELLS$|$
CELLS.down ≥ 0
CELLS.down $\leq |$CELLS$|$

Purpose

Enforce all cells of the collection CELLS to be connected. Each cell is defined by the following attributes:

1. The index attribute of the cell, which is an integer between 1 and the total number of cells, is unique for each cell.
2. The right 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 left 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 up attribute, which is the index of the cell located immediately on top of that cell (or 0 if no such cell exists).
5. The down attribute, which is the index of the cell located immediately above that cell (or 0 if no such cell exists).

This corresponds to a polyomino [118].

Arc input(s) CELLS

Arc generator $CLIQUE(\neq) \mapsto$ 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)

- $\mathbf{NVERTEX} = |\texttt{CELLS}|$
- $\mathbf{NCC} = 1$

Example

$$\texttt{polyomino}\left(\left\{\begin{array}{lllll}\texttt{index}-1 & \texttt{right}-0 & \texttt{left}-0 & \texttt{up}-2 & \texttt{down}-0,\\ \texttt{index}-2 & \texttt{right}-3 & \texttt{left}-0 & \texttt{up}-0 & \texttt{down}-1,\\ \texttt{index}-3 & \texttt{right}-0 & \texttt{left}-2 & \texttt{up}-4 & \texttt{down}-0,\\ \texttt{index}-4 & \texttt{right}-5 & \texttt{left}-0 & \texttt{up}-0 & \texttt{down}-3,\\ \texttt{index}-5 & \texttt{right}-0 & \texttt{left}-4 & \texttt{up}-0 & \texttt{down}-0\end{array}\right\}\right)$$

Parts (A) and (B) of Figure 4.351 respectively show the initial and final graph. Since we use the $\mathbf{NVERTEX}$ graph property the vertices of the final graph are stressed in bold. Since we also use the $\mathbf{NCC}$ 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.

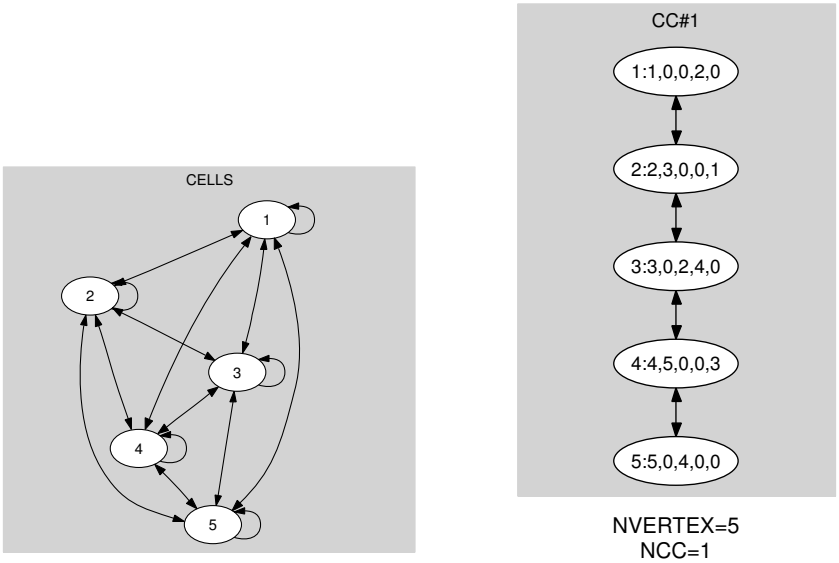


Figure 4.351: Initial and final graph of the polyomino constraint

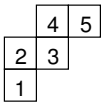


Figure 4.352: Polyomino corresponding to the final graph

Graph model

The graph constraint models the fact that all the cells are connected. We use the $CLIQUE(\neq)$ arc generator in order to only consider connections between two distinct cells. The first graph property $\mathbf{NVERTEX} = |\texttt{CELLS}|$ 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 $\mathbf{NVERTEX} = |\texttt{CELLS}|$ and from the restriction $|\texttt{CELLS}| \geq 1$ 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 $\overline{\underline{\mathbf{NCC}}}$ to $\underline{\mathbf{NCC}}$.

Usage Enumeration of polyominoes.

Key words geometrical constraint, strongly connected component, pentomino.

4.171 product_ctr

Origin Arithmetic constraint.

Constraint product_ctr(VARIABLES, CTR, VAR)

Argument(s)

VARIABLES	:	collection(var − dvar)
CTR	:	atom
VAR	:	dvar

Restriction(s) required(VARIABLES, var)
CTR $\in [=, \neq, <, \geq, >, \leq]$

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 $SELF \mapsto$ collection(variables)

Arc arity 1

Arc constraint(s) TRUE

Graph property(ies) $PRODUCT$(VARIABLES, var) CTR VAR

Example product_ctr($\{$var − 2, var − 1, var − 4$\}, =, 8$)

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 $SELF$ arc generator together with the TRUE arc constraint. This predefined arc constraint allways holds.

Used in cumulative_product.

See also sum_ctr, range_ctr.

Key words arithmetic constraint, product.

4.172 range_ctr

Origin Arithmetic constraint.

Constraint range_ctr(VARIABLES, CTR, VAR)

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 CTR VAR.

Arc input(s) VARIABLES

Arc generator $SELF \mapsto$ 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.

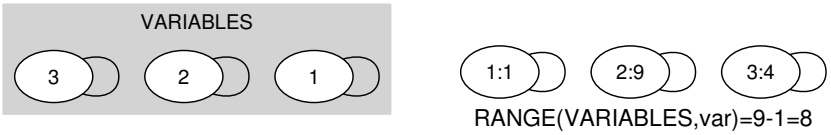


Figure 4.354: Initial and final graph of the range_ctr constraint

Graph model Since we want to keep all the vertices of the initial graph we use the $SELF$ arc generator together with the TRUE arc constraint. This predefined arc constraint allways holds.

Used in shift.

See also sum_ctr, product_ctr.

Key words arithmetic constraint, range.

4.173 relaxed_sliding_sum

Origin CHIP

Constraint relaxed_sliding_sum(ATLEAST, ATMOST, LOW, UP, SEQ, VARIABLES)

Argument(s)

ATLEAST	:	int
ATMOST	:	int
LOW	:	int
UP	:	int
SEQ	:	int
VARIABLES	:	collection(var − dvar)

Restriction(s)
- ATLEAST ≥ 0
- ATMOST $\geq$ ATLEAST
- ATMOST $\leq |\text{VARIABLES}| - \text{SEQ} + 1$
- UP $\geq$ LOW
- SEQ > 0
- SEQ $\leq |\text{VARIABLES}|$
- required(VARIABLES, var)

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 $PATH \mapsto$ collection

Arc arity SEQ

Arc constraint(s)
- sum_ctr(collection, $\geq$, LOW)
- sum_ctr(collection, $\leq$, UP)

Graph property(ies)
- **NARC** $\geq$ ATLEAST
- **NARC** $\leq$ ATMOST

Example

$$\text{relaxed\_sliding\_sum}\left(\begin{array}{c} 3,4,3,7,4, \\ \left\{\begin{array}{l} \text{var}-2, \\ \text{var}-4, \\ \text{var}-2, \\ \text{var}-0, \\ \text{var}-0, \\ \text{var}-3, \\ \text{var}-4 \end{array}\right\} \end{array}\right)$$

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.

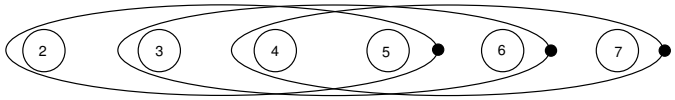


Figure 4.355: Final directed hypergraph associated to the example

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 same(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 VARIABLES2 collection correspond to the variables of the VARIABLES1 collection according to a permutation.

Arc input(s) VARIABLES1 VARIABLES2

Arc generator $PRODUCT \mapsto$ collection(variables1, variables2)

Arc arity 2

Arc constraint(s) variables1.var = variables2.var

Graph property(ies)
- for all connected components: $\mathbf{NSOURCE} = \mathbf{NSINK}$
- $\mathbf{NSOURCE} =$ |VARIABLES1|
- $\mathbf{NSINK} =$ |VARIABLES2|

Example

$$\texttt{same}\left(\left\{\begin{array}{l}\texttt{var}-1,\\ \texttt{var}-9,\\ \texttt{var}-1,\\ \texttt{var}-5,\\ \texttt{var}-2,\\ \texttt{var}-1\end{array}\right\},\left\{\begin{array}{l}\texttt{var}-9,\\ \texttt{var}-1,\\ \texttt{var}-1,\\ \texttt{var}-1,\\ \texttt{var}-2,\\ \texttt{var}-5\end{array}\right\}\right)$$

Parts (A) and (B) of Figure 4.356 respectively show the initial and final graph. Since we use the $\mathbf{NSOURCE}$ and $\mathbf{NSINK}$ 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 |VARIABLES1|.
- The number of sinks of the final graph is equal to |VARIABLES2|.

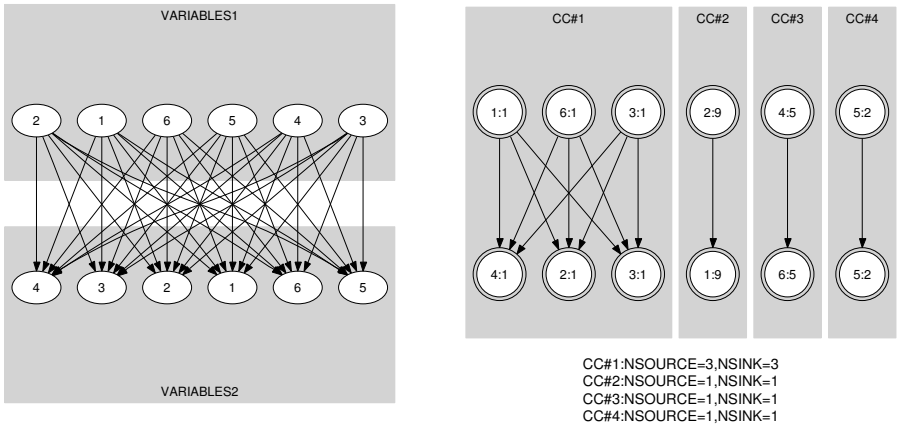


Figure 4.356: Initial and final graph of the same constraint

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 |VARIABLES1| and |VARIABLES2|. Therefore we can rewrite $\mathbf{NSOURCE} = |\texttt{VARIABLES1}|$ to $\mathbf{NSOURCE} \geq |\texttt{VARIABLES1}|$ and simplify $\overline{\mathbf{NSOURCE}}$ to $\overline{\mathbf{NSOURCE}}$. In a similar way, we can rewrite $\mathbf{NSINK} = |\texttt{VARIABLES2}|$ to $\mathbf{NSINK} \geq |\texttt{VARIABLES2}|$ and simplify $\overline{\mathbf{NSINK}}$ to $\overline{\mathbf{NSINK}}$.

Automaton

To each item of the collection VARIABLES1 corresponds a signature variable $\texttt{S}_i$, which is equal to 0. To each item of the collection VARIABLES2 corresponds a signature variable $\texttt{S}_{i+|\texttt{VARIABLES1}|}$, 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`$(D = d_1, d_2, \ldots, d_m, N = n_1, n_2, \ldots, n_m)$ constraint where each doctor is respresented 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 $v_{p,1}, v_{p,2}, \ldots, v_{p,D}$. $v_{p,D}$ 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 $v_{1,1}, v_{1,2}, \ldots, v_{1,D}$ should be a permutation of $v_{p,1}, v_{p,2}, \ldots, v_{p,D}$ for each $(1 < p \leq P)$.
- The `same` constraint can also be used as a chanelling 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 ith line and the jth 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 $k \in [1, c_i]$. 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

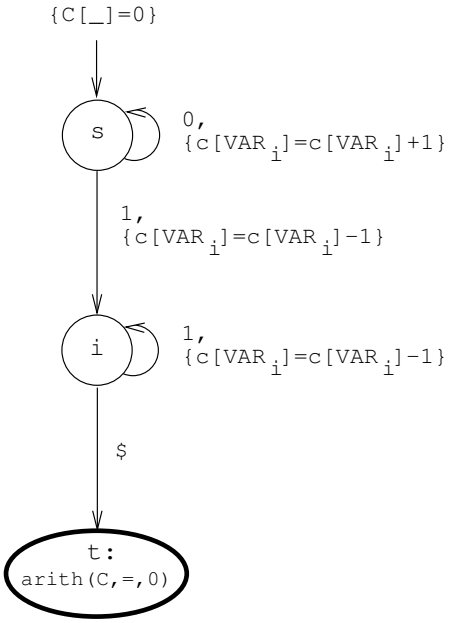


Figure 4.357: Automaton of the `same` constraint

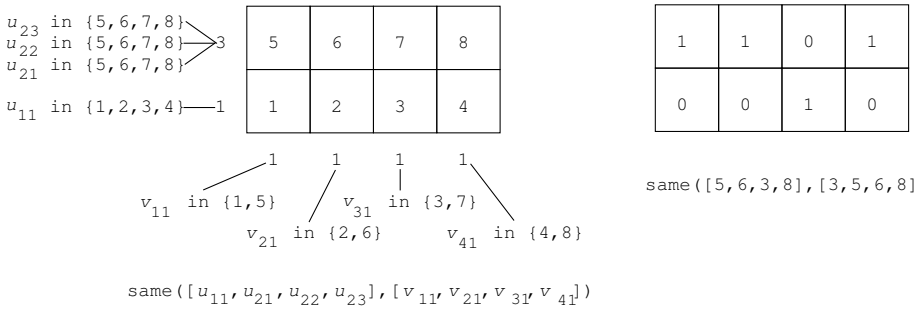


Figure 4.358: Modelling the 0-1 matrix reconstruction problem with the `same` constraint

$$\texttt{same}\left(\begin{array}{l}\left\{\begin{array}{l}\texttt{var}-x_1,\texttt{var}-x_2\end{array}\right\},\\ \left\{\begin{array}{l}\texttt{var}-y_1,\texttt{var}-y_2\end{array}\right\},\end{array}\right)$$

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:

$$\texttt{global\_cardinality}\left(\begin{array}{l}\left\{\begin{array}{l}\texttt{var}-x_1,\texttt{var}-x_2\end{array}\right\},\\ \left\{\begin{array}{ll}\texttt{val}-1 & \texttt{noccurrence}-c_1,\\ \texttt{val}-2 & \texttt{noccurrence}-c_2,\\ \texttt{val}-3 & \texttt{noccurrence}-c_3,\\ \texttt{val}-4 & \texttt{noccurrence}-c_4\end{array}\right\}\end{array}\right)$$

$$\texttt{global\_cardinality}\left(\begin{array}{l}\left\{\begin{array}{l}\texttt{var}-y_1,\texttt{var}-y_2\end{array}\right\},\\ \left\{\begin{array}{ll}\texttt{val}-1 & \texttt{noccurrence}-c_1,\\ \texttt{val}-2 & \texttt{noccurrence}-c_2,\\ \texttt{val}-3 & \texttt{noccurrence}-c_3,\\ \texttt{val}-4 & \texttt{noccurrence}-c_4\end{array}\right\}\end{array}\right)$$

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.

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 ≥ 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 $PRODUCT \mapsto$ collection(variables1, variables2)

Arc arity 2

Arc constraint(s) variables1.var = variables2.var

Graph property(ies)
- for all connected components: **NSOURCE** = **NSINK**
- **NSOURCE** = |VARIABLES1|
- **NSINK** = |VARIABLES2|

For all items of VALUES:

Arc input(s) VARIABLES1

Arc generator $SELF \mapsto$ collection(variables)

Arc arity 1

Arc constraint(s) variables.var = VALUES.val

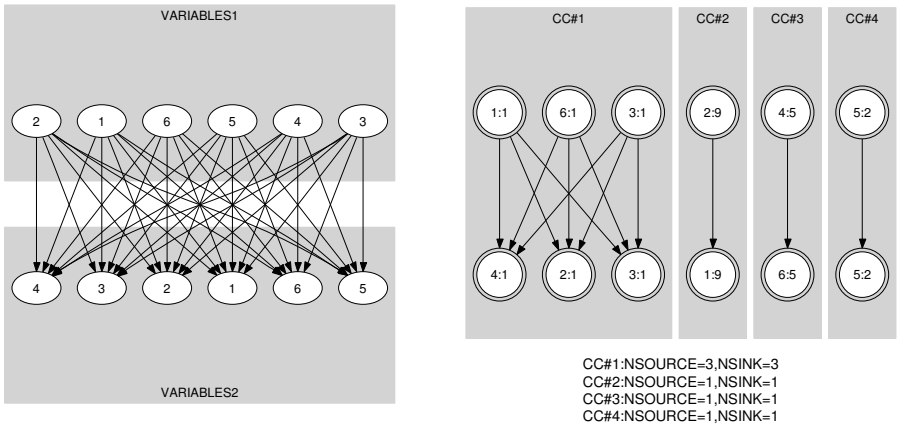
Graph property(ies) **NVERTEX** = VALUES.noccurrence

Example

$$\texttt{same\_and\_global\_cardinality}\left(\left\{\begin{array}{l}\texttt{var}-1,\\ \texttt{var}-9,\\ \texttt{var}-1,\\ \texttt{var}-5,\\ \texttt{var}-2,\\ \texttt{var}-1\end{array}\right\},\left\{\begin{array}{l}\texttt{var}-9,\\ \texttt{var}-1,\\ \texttt{var}-1,\\ \texttt{var}-1,\\ \texttt{var}-2,\\ \texttt{var}-5\end{array}\right\},\left\{\begin{array}{ll}\texttt{val}-1 & \texttt{noccurrence}-3,\\ \texttt{val}-2 & \texttt{noccurrence}-1,\\ \texttt{val}-5 & \texttt{noccurrence}-1,\\ \texttt{val}-7 & \texttt{noccurrence}-0,\\ \texttt{val}-9 & \texttt{noccurrence}-1\end{array}\right\}\right)$$

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) **(B)**

Figure 4.359: Initial and final graph of the `same_and_global_cardinality` constraint

762 $\overline{\mathbf{NSINK}}$, $\overline{\mathbf{NSOURCE}}$, CC($\overline{\mathbf{NSINK}}$, $\overline{\mathbf{NSOURCE}}$), $PRODUCT$; $\overline{\mathbf{NVERTEX}}$, $SELF$, $\forall$

Usage The same_and_global_cardinality 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 same constraint [141, 25] is extended to support the cardinalities. Then, algorithms are developed to compute arc-consistency and bound-consistency.

See also same, global_cardinality.

Key words constraint between two collections of variables, value constraint, permutation, multiset, equality between multisets, assignment, demand profile.

4.176 same_intersection

Origin Derived from same and common.

Constraint same_intersection(VARIABLES1, VARIABLES2)

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 $PRODUCT \mapsto$ collection(variables1, variables2)

Arc arity 2

Arc constraint(s) variables1.var = variables2.var

Graph property(ies) for all connected components: $\mathbf{NSOURCE} = \mathbf{NSINK}$

Example

$$\texttt{same\_intersection}\left(\left\{\begin{array}{l}\texttt{var}-1,\\ \texttt{var}-9,\\ \texttt{var}-1,\\ \texttt{var}-5,\\ \texttt{var}-2,\\ \texttt{var}-1\end{array}\right\},\left\{\begin{array}{l}\texttt{var}-9,\\ \texttt{var}-1,\\ \texttt{var}-1,\\ \texttt{var}-1,\\ \texttt{var}-3,\\ \texttt{var}-5,\\ \texttt{var}-8\end{array}\right\}\right)$$

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.

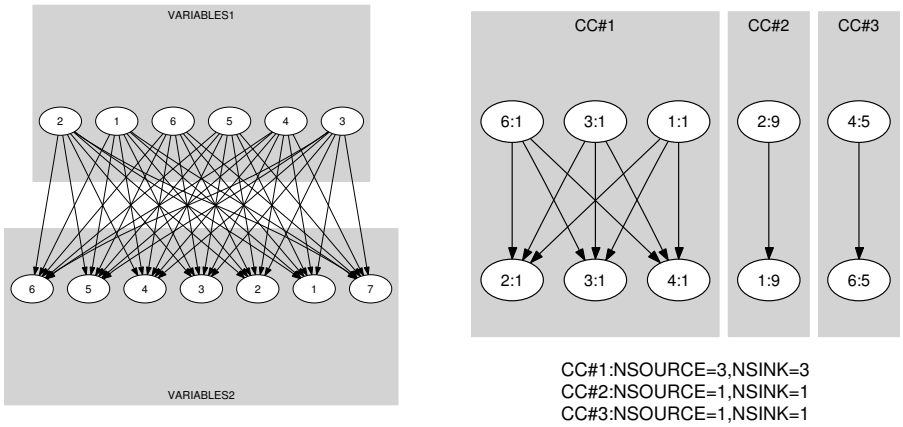


Figure 4.360: Initial and final graph of the `same_intersection` constraint

4.177 same_interval

Origin Derived from same.

Constraint same_interval(VARIABLES1, VARIABLES2, SIZE_INTERVAL)

Argument(s)

VARIABLES1	:	collection(var − dvar)
VARIABLES2	:	collection(var − dvar)
SIZE_INTERVAL	:	int

Restriction(s)
- |VARIABLES1| = |VARIABLES2|
- required(VARIABLES1, var)
- required(VARIABLES2, var)
- 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 [SIZE_INTERVAL $\cdot\, i$, SIZE_INTERVAL $\cdot\, i +$ SIZE_INTERVAL $- 1$. For all integer i we have $N_i = M_i$.

Arc input(s) VARIABLES1 VARIABLES2

Arc generator $PRODUCT \mapsto$ collection(variables1, variables2)

Arc arity 2

Arc constraint(s) variables1.var/SIZE_INTERVAL = variables2.var/SIZE_INTERVAL

Graph property(ies)
- for all connected components: $\mathbf{NSOURCE} = \mathbf{NSINK}$
- $\mathbf{NSOURCE} =$ |VARIABLES1|
- $\mathbf{NSINK} =$ |VARIABLES2|

Example

$$\texttt{same\_interval}\left(\left\{\begin{array}{l}\texttt{var}-1,\\ \texttt{var}-7,\\ \texttt{var}-6,\\ \texttt{var}-0,\\ \texttt{var}-1,\\ \texttt{var}-7\end{array}\right\},\left\{\begin{array}{l}\texttt{var}-8,\\ \texttt{var}-8,\\ \texttt{var}-8,\\ \texttt{var}-0,\\ \texttt{var}-1,\\ \texttt{var}-2\end{array}\right\},3\right)$$

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 4.361 respectively show the initial and final graph. Since we use the $\mathbf{NSOURCE}$ and $\mathbf{NSINK}$ 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 |`VARIABLES1`|.
- The number of sinks of the final graph is equal to |`VARIABLES2`|.

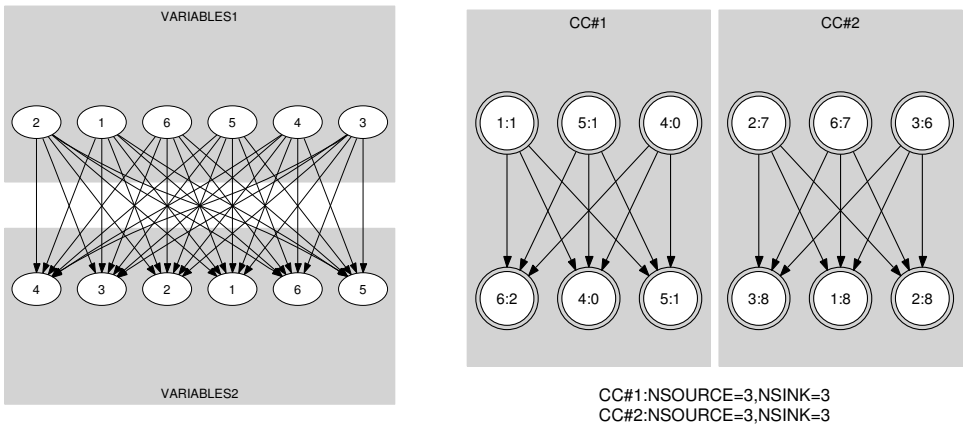


Figure 4.361: Initial and final graph of the `same_interval` constraint

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 |`VARIABLES1`| and |`VARIABLES2`|. Therefore we can rewrite $\mathbf{NSOURCE} = |\texttt{VARIABLES1}|$ to $\mathbf{NSOURCE} \geq |\texttt{VARIABLES1}|$ and simplify $\overline{\mathbf{NSOURCE}}$ to $\overline{\mathbf{NSOURCE}}$. In a similar way, we can rewrite $\mathbf{NSINK} = |\texttt{VARIABLES2}|$ to $\mathbf{NSINK} \geq |\texttt{VARIABLES2}|$ and simplify $\overline{\mathbf{NSINK}}$ to $\overline{\mathbf{NSINK}}$.

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 same_modulo(VARIABLES1, VARIABLES2, M)

Argument(s)

VARIABLES1	:	collection(var − dvar)
VARIABLES2	:	collection(var − dvar)
M	:	int

Restriction(s)
- |VARIABLES1| = |VARIABLES2|
- required(VARIABLES1, var)
- required(VARIABLES2, var)
- M > 0

Purpose For each integer R in $[0, \mathtt{M}-1]$, 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 $[0, \mathtt{M}-1]$ we have that $N1_R = N2_R$.

Arc input(s) VARIABLES1 VARIABLES2

Arc generator $PRODUCT \mapsto$ collection(variables1, variables2)

Arc arity 2

Arc constraint(s) variables1.var mod M = variables2.var mod M

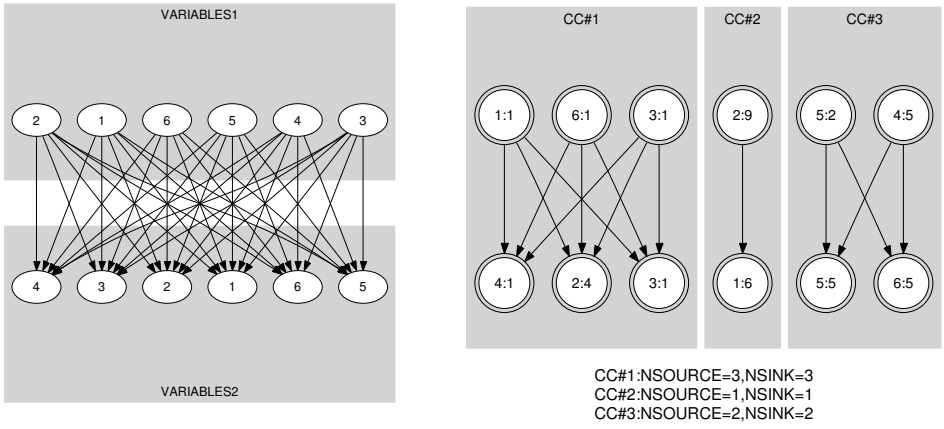
Graph property(ies)
- for all connected components: **NSOURCE** = **NSINK**
- **NSOURCE** = |VARIABLES1|
- **NSINK** = |VARIABLES2|

Example

$$\mathtt{same\_modulo}\left(\left\{\begin{array}{l}\mathtt{var}-1,\\ \mathtt{var}-9,\\ \mathtt{var}-1,\\ \mathtt{var}-5,\\ \mathtt{var}-2,\\ \mathtt{var}-1\end{array}\right\},\left\{\begin{array}{l}\mathtt{var}-6,\\ \mathtt{var}-4,\\ \mathtt{var}-1,\\ \mathtt{var}-1,\\ \mathtt{var}-5,\\ \mathtt{var}-5\end{array}\right\},3\right)$$

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 |VARIABLES1|.
- The number of sinks of the final graph is equal to |VARIABLES2|.



(A) **(B)**

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 |VARIABLES1| and |VARIABLES2|. Therefore we can rewrite $\mathbf{NSOURCE} = |\texttt{VARIABLES1}|$ to $\mathbf{NSOURCE} \geq |\texttt{VARIABLES1}|$ and simplify $\underline{\mathbf{NSOURCE}}$ to $\overline{\mathbf{NSOURCE}}$. In a similar way, we can rewrite $\mathbf{NSINK} = |\texttt{VARIABLES2}|$ to $\mathbf{NSINK} \geq |\texttt{VARIABLES2}|$ and simplify $\underline{\mathbf{NSINK}}$ to $\overline{\mathbf{NSINK}}$.

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| ≥ 2

Purpose For each integer i in $[1, |\texttt{PARTITIONS}|]$, 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 $[1, |\texttt{PARTITIONS}|]$ we have $N1_i = N2_i$.

Arc input(s) VARIABLES1 VARIABLES2

Arc generator $PRODUCT \mapsto$ collection(variables1, variables2)

Arc arity 2

Arc constraint(s) in_same_partition(variables1.var, variables2.var, PARTITIONS)

Graph property(ies)
- for all connected components: $\mathbf{NSOURCE} = \mathbf{NSINK}$
- $\mathbf{NSOURCE} = |\texttt{VARIABLES1}|$
- $\mathbf{NSINK} = |\texttt{VARIABLES2}|$

Example

$$\texttt{same}\left(\left\{\begin{array}{l}\texttt{var}-1,\\ \texttt{var}-2,\\ \texttt{var}-6,\\ \texttt{var}-3,\\ \texttt{var}-1,\\ \texttt{var}-2\end{array}\right\},\left\{\begin{array}{l}\texttt{var}-6,\\ \texttt{var}-6,\\ \texttt{var}-2,\\ \texttt{var}-3,\\ \texttt{var}-1,\\ \texttt{var}-3\end{array}\right\},\left\{\begin{array}{l}\texttt{p}-\{\texttt{val}-1,\texttt{val}-3\},\\ \texttt{p}-\{\texttt{val}-4\},\\ \texttt{p}-\{\texttt{val}-2,\texttt{val}-6\}\end{array}\right\}\right)$$

Parts (A) and (B) of Figure 4.363 respectively show the initial and final graph. Since we use the $\mathbf{NSOURCE}$ and $\mathbf{NSINK}$ 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 |`VARIABLES1`|.
- The number of sinks of the final graph is equal to |`VARIABLES2`|.

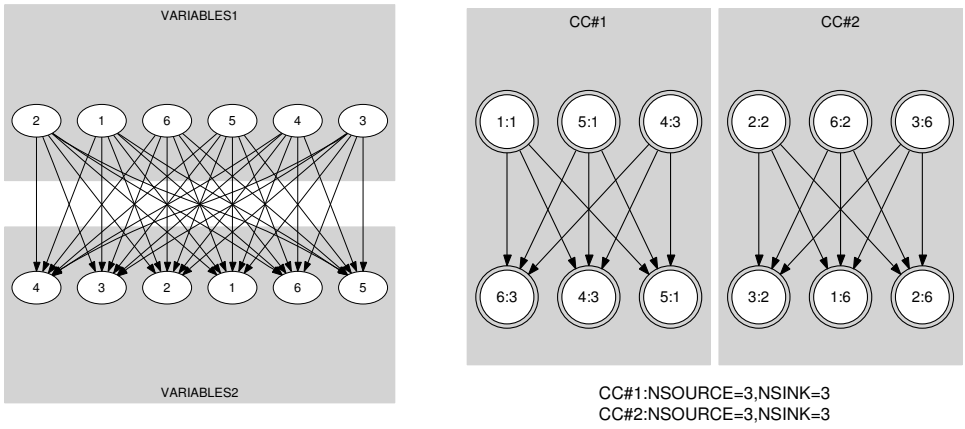


Figure 4.363: Initial and final graph of the `same_partition` constraint

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 |`VARIABLES1`| and |`VARIABLES2`|. Therefore we can rewrite $\mathbf{NSOURCE} = |\texttt{VARIABLES1}|$ to $\mathbf{NSOURCE} \geq |\texttt{VARIABLES1}|$ and simplify $\underline{\mathbf{NSOURCE}}$ to $\overline{\mathbf{NSOURCE}}$. In a similar way, we can rewrite $\mathbf{NSINK} = |\texttt{VARIABLES2}|$ to $\mathbf{NSINK} \geq |\texttt{VARIABLES2}|$ and simplify $\underline{\mathbf{NSINK}}$ to $\overline{\mathbf{NSINK}}$.

See also `same`, `in_same_partition`.

Key words constraint between two collections of variables, permutation, partition.

4.180 sequence_folding

Origin J. Pearson

Constraint sequence_folding(LETTERS)

Argument(s) LETTERS : collection(index − int, next − dvar)

Restriction(s) |LETTERS| ≥ 1
required(LETTERS, [index, next])
LETTERS.index ≥ 1
LETTERS.index ≤ |LETTERS|
increasing_seq(LETTERS, index)
LETTERS.next ≥ 1
LETTERS.next ≤ |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 $SELF \mapsto$ collection(letters)

Arc arity 1

Arc constraint(s) letters.next ≥ letters.index

Graph property(ies) $\mathbf{NARC} = |\text{LETTERS}|$

Arc input(s) LETTERS

Arc generator $CLIQUE(<) \mapsto$ collection(letters1, letters2)

Arc arity 2

Arc constraint(s) letters2.index ≥ letters1.next ∨ letters2.next ≤ letters1.next

Graph property(ies) $\mathbf{NARC} = |\text{LETTERS}| * (|\text{LETTERS}| - 1)/2$

Example

$$\text{sequence\_folding}\left(\left\{\begin{array}{ll}\text{index}-1 & \text{next}-1,\\ \text{index}-2 & \text{next}-8,\\ \text{index}-3 & \text{next}-3,\\ \text{index}-4 & \text{next}-5,\\ \text{index}-5 & \text{next}-5,\\ \text{index}-6 & \text{next}-7,\\ \text{index}-7 & \text{next}-7,\\ \text{index}-8 & \text{next}-8,\\ \text{index}-9 & \text{next}-9\end{array}\right\}\right)$$

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.

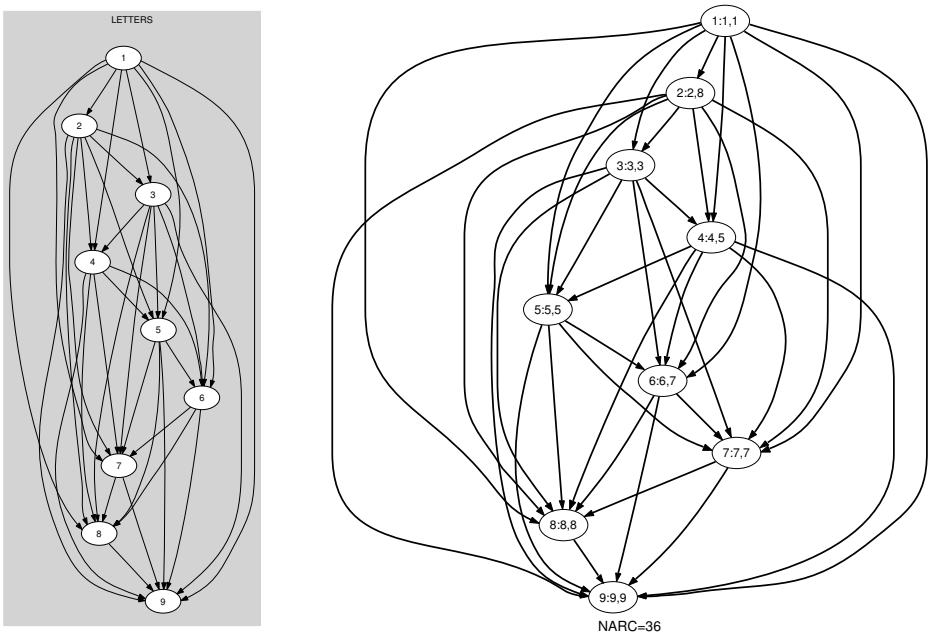


Figure 4.364: Initial and final graph of the `sequence_folding` constraint

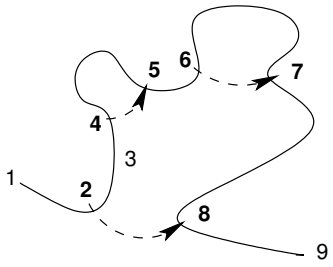


Figure 4.365: Folded sequence associated to the example

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 |LETTERS|. Therefore we can rewrite the graph property $\mathbf{NARC} = |\texttt{LETTERS}|$ to $\mathbf{NARC} \geq |\texttt{LETTERS}|$ and simplify $\underline{\overline{\mathbf{NARC}}}$ to $\overline{\mathbf{NARC}}$.

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 $|\texttt{LETTERS}| \cdot (|\texttt{LETTERS}| - 1)/2$. Therefore we can rewrite the graph property $\mathbf{NARC} = |\texttt{LETTERS}| \cdot (|\texttt{LETTERS}| - 1)/2$ to $\mathbf{NARC} \geq |\texttt{LETTERS}| \cdot (|\texttt{LETTERS}| - 1)/2$ and simplify $\underline{\overline{\mathbf{NARC}}}$ to $\overline{\mathbf{NARC}}$.

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 $\texttt{INDEX}_i$ and $\texttt{NEXT}_i$ respectively denote the index and the next attributes of the i^{th} item of the collection LETTERS. Similarly, let $\texttt{INDEX}_j$ and $\texttt{NEXT}_j$ respectively denote the index and the next attributes of the j^{th} item of the collection LETTERS. To each quadruple $(\texttt{INDEX}_i, \texttt{NEXT}_i, \texttt{INDEX}_j, \texttt{NEXT}_j)$ corresponds a signature variable $\texttt{S}_{i,j}$, which takes its value in $\{0, 1, 2\}$, as well as the following signature constraint:

$(\texttt{INDEX}_i \leq \texttt{NEXT}_i) \wedge (\texttt{INDEX}_j \leq \texttt{NEXT}_j) \wedge (\texttt{NEXT}_i \leq \texttt{NEXT}_j) \Leftrightarrow \texttt{S}_{i,j} = 0 \wedge$

$(\texttt{INDEX}_i \leq \texttt{NEXT}_i) \wedge (\texttt{INDEX}_j \leq \texttt{NEXT}_j) \wedge (\texttt{NEXT}_i > \texttt{INDEX}_j) \wedge (\texttt{NEXT}_j \leq \texttt{NEXT}_i) \Leftrightarrow \texttt{S}_{i,j} = 1.$

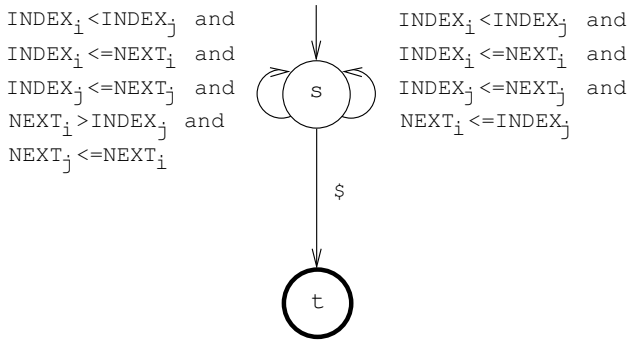


Figure 4.366: Automaton of the sequence_folding constraint

Usage

Motivated by RNA folding [144].

Key words

decomposition, geometrical constraint, sequence, bioinformatics, automaton, automaton without counters.

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 ≠ 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

$$\texttt{set\_value\_precede}\left(\begin{array}{c} 2, 1, \left\{\begin{array}{l} \texttt{var} - \{0, 2\}, \\ \texttt{var} - \{0, 1\}, \\ \texttt{var} - \emptyset, \\ \texttt{var} - \{1\} \end{array}\right\} \end{array}\right)$$

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.

4.182 shift

Origin N. Beldiceanu

Constraint shift(MIN_BREAK, MAX_RANGE, TASKS)

Argument(s)

MIN_BREAK	:	int
MAX_RANGE	:	int
TASKS	:	collection(id − int, origin − dvar, end − dvar)

Restriction(s)
- MIN_BREAK > 0
- MAX_RANGE > 0
- required(TASKS, [id, origin, end])
- distinct(TASKS, id)

Purpose

The difference between the end of the last task of a *shift* and the origin of the first task of a *shift* should not exceed the quantity MAX_RANGE. Two tasks t_1 and t_2 belong to the *same shift* if at least one of the following conditions is true:
- Task t_2 starts after the end of task t_1 at a distance that is less than or equal to the quantity MIN_BREAK,
- Task t_1 starts after the end of task t_2 at a distance that is less than or equal to the quantity MIN_BREAK.
- Task t_1 overlaps task t_2.

Arc input(s) TASKS

Arc generator $SELF \mapsto$ collection(tasks)

Arc arity 1

Arc constraint(s)
- tasks.end $\geq$ tasks.origin
- tasks.end − tasks.origin $\leq$ MAX_RANGE

Graph property(ies) $\mathbf{NARC} = |\text{TASKS}|$

Arc input(s) TASKS

Arc generator $CLIQUE \mapsto$ collection(tasks1, tasks2)

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

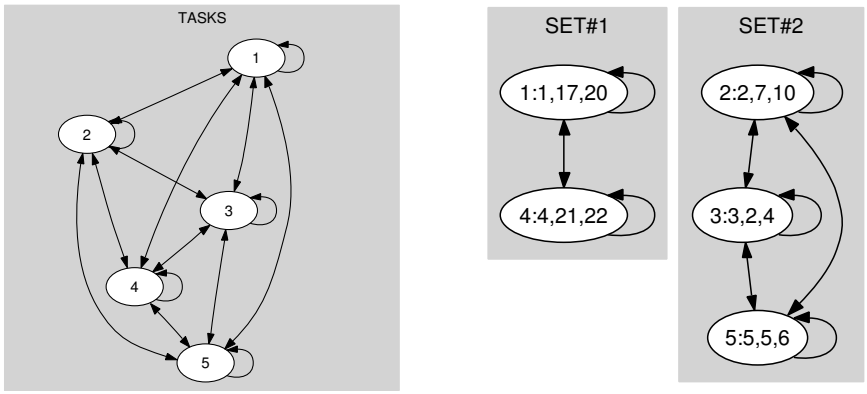
$$\text{CC} \mapsto \left[\text{variables} - \text{col} \left(\begin{array}{l} \text{VARIABLES} - \text{collection}(\text{var} - \text{dvar}), \\ [\text{item}(\text{var} - \text{TASKS.origin}), \text{item}(\text{var} - \text{TASKS.end})] \end{array} \right) \right]$$

Constraint(s) on sets `range_ctr`$(\texttt{variables}, \leq, \texttt{MAX\_RANGE})$

Example

$$\texttt{shift}\left(\begin{array}{l} 6, 8, \left\{\begin{array}{lll} \texttt{id}-1 & \texttt{origin}-17 & \texttt{end}-20, \\ \texttt{id}-2 & \texttt{origin}-7 & \texttt{end}-10, \\ \texttt{id}-3 & \texttt{origin}-2 & \texttt{end}-4, \\ \texttt{id}-4 & \texttt{origin}-21 & \texttt{end}-22, \\ \texttt{id}-5 & \texttt{origin}-5 & \texttt{end}-6 \end{array}\right\} \end{array}\right)$$

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 consecutives 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).



(A) **(B)**

Figure 4.367: Initial and final graph of the `shift` constraint

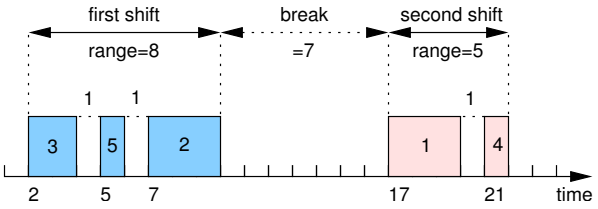


Figure 4.368: The two shifts of the example

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 |TASKS|. Therefore we can rewrite the graph property $\mathbf{NARC} = |\texttt{TASKS}|$ to $\mathbf{NARC} \geq |\texttt{TASKS}|$ and simplify $\underline{\overline{\mathbf{NARC}}}$ to $\overline{\mathbf{NARC}}$.

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.

4.183 size_maximal_sequence_alldifferent

Origin N. Beldiceanu

Constraint size_maximal_sequence_alldifferent(SIZE, VARIABLES)

Synonym(s) size_maximal_sequence_alldiff, size_maximal_sequence_alldistinct.

Argument(s)

SIZE	:	dvar
VARIABLES	:	collection(var − dvar)

Restriction(s)

SIZE ≥ 0
SIZE $\leq |$VARIABLES$|$
required(VARIABLES, var)

Purpose SIZE is the size of the maximal sequence (among all sequences of consecutives variables of the collection VARIABLES) for which the alldifferent constraint holds.

Arc input(s) VARIABLES

Arc generator $PATH\_N \mapsto$ collection

Arc arity $*$

Arc constraint(s) alldifferent(collection)

Graph property(ies) NARC = SIZE

Example size_maximal_sequence_alldifferent$\left(4, \left\{ \begin{array}{l} \mathtt{var}-2, \\ \mathtt{var}-2, \\ \mathtt{var}-4, \\ \mathtt{var}-5, \\ \mathtt{var}-2, \\ \mathtt{var}-7, \\ \mathtt{var}-4 \end{array} \right\} \right)$

The previous constraint holds since the constraint alldifferent(var − 4, var − 5, var − 2, var − 7) holds and since the following three constraints do not hold:

- alldifferent(var − 2, var − 2, var − 4, var − 5, var − 2),
- alldifferent(var − 2, var − 4, var − 5, var − 2, var − 7),
- alldifferent(var − 4, var − 5, var − 2, var − 7, var − 4).

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 alldifferent, size_maximal_starting_sequence_alldifferent.

Key words sliding sequence constraint, conditional constraint, sequence, hypergraph.

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 consecutives variables of the collection VARIABLES starting at position one) for which the alldifferent constraint holds.

Arc input(s) VARIABLES

Arc generator $PATH\_1 \mapsto$ collection

Arc arity $*$

Arc constraint(s) alldifferent(collection)

Graph property(ies) NARC = SIZE

Example size_maximal_starting_sequence_alldifferent $\left\langle 4, \left\{ \begin{array}{l} \mathtt{var}-9, \\ \mathtt{var}-2, \\ \mathtt{var}-4, \\ \mathtt{var}-5, \\ \mathtt{var}-2, \\ \mathtt{var}-7, \\ \mathtt{var}-4 \end{array} \right\} \right\rangle$

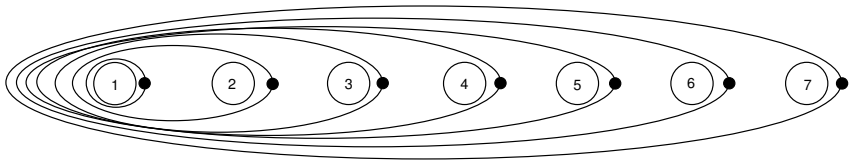
The previous constraint holds since the constraint alldifferent(var − 9, var − 2, var − 4, var − 5) holds and since alldifferent(var − 9, var − 2, var − 4, var − 5, var − 2) does not hold. Parts (A) and (B) of Figure 4.369 respectively show the initial and final graph.

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.

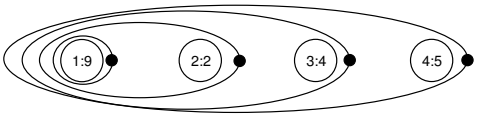
Remark A *conditional constraint* [145] with the specific structure that one can relax the constraints on the last variables of the collection VARIABLES.

See also alldifferent, size_maximal_sequence_alldifferent.

Key words sliding sequence constraint, conditional constraint, sequence, hypergraph.



(A)



(B)

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

Let n be the total number of variables of the collection VARIABLES. A *maximum non-zero set of consecutive variables* $X_i..X_j (1 \leq i \leq j \leq n)$ is defined in the following way:
- All variables $X_i, \ldots, X_j$ 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.

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.

Arc input(s) VARIABLES

Arc generator
- $PATH \mapsto$ collection(variables1, variables2)
- $LOOP \mapsto$ collection(variables1, variables2)

Arc arity 2

Arc constraint(s)
- variables1.var $\neq 0$
- variables2.var $\neq 0$

Sets CC $\mapsto$ [variables]

Constraint(s) on sets among_low_up(ATLEAST, ATMOST, variables, VALUES)

Example

$$\text{sliding\_card\_skip0}\left(\left\langle \begin{array}{c} 2, 3, \left\{ \begin{array}{l} \text{var} - 0, \\ \text{var} - 7, \\ \text{var} - 2, \\ \text{var} - 9, \\ \text{var} - 0, \\ \text{var} - 0, \\ \text{var} - 9, \\ \text{var} - 4, \\ \text{var} - 9 \end{array} \right\}, \\ \{\text{val} - 7, \text{val} - 9\} \end{array} \right\rangle\right)$$

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.

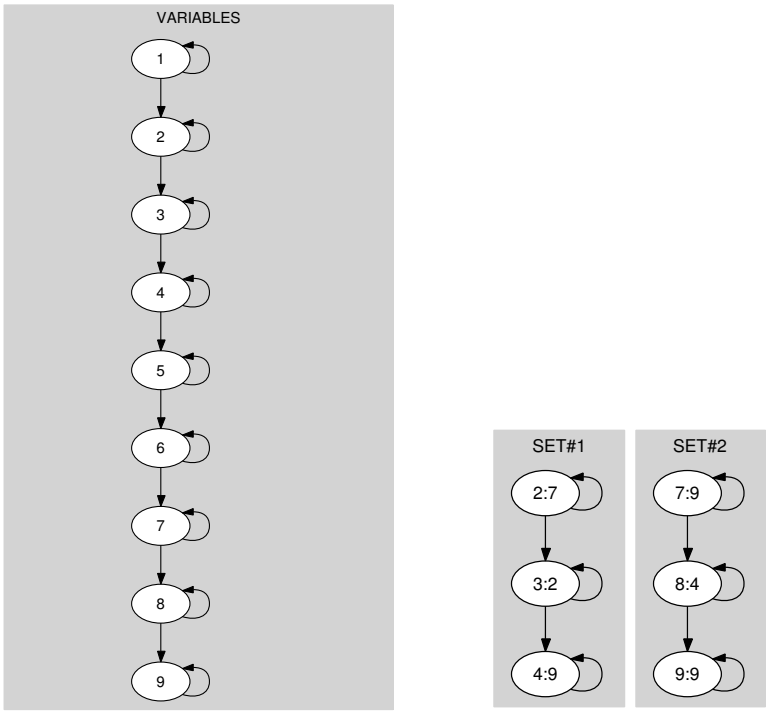


Figure 4.370: Initial and final graph of the `sliding_card_skip0` constraint

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 $\texttt{VAR}_i$ of the collection `VARIABLES` corresponds a signature variable $\texttt{S}_i$. The following signature constraint links $\texttt{VAR}_i$ and $\texttt{S}_i$:

$$(\texttt{VAR}_i = 0) \Leftrightarrow \texttt{S}_i = 0 \wedge$$
$$(\texttt{VAR}_i \neq 0 \wedge \texttt{VAR}_i \notin \texttt{VALUES}) \Leftrightarrow \texttt{S}_i = 1 \wedge$$
$$(\texttt{VAR}_i \neq 0 \wedge \texttt{VAR}_i \in \texttt{VALUES}) \Leftrightarrow \texttt{S}_i = 2.$$

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.

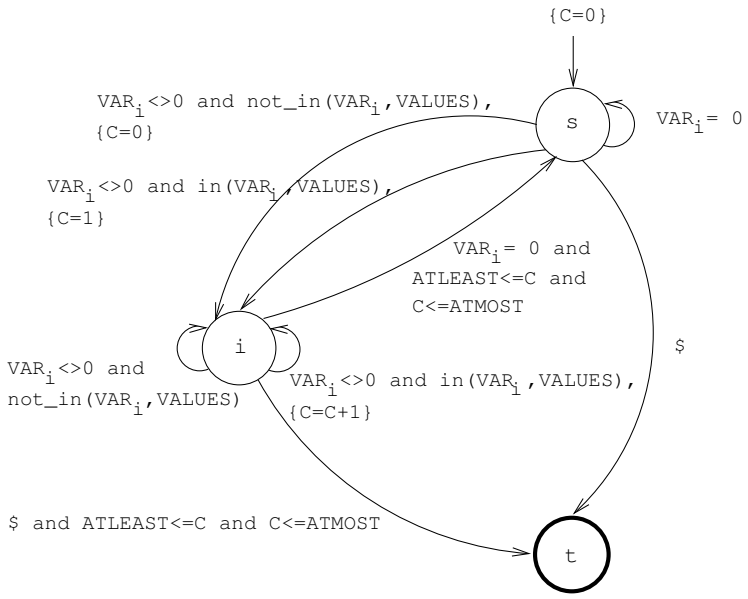


Figure 4.371: Automaton of the `sliding_card_skip0` constraint

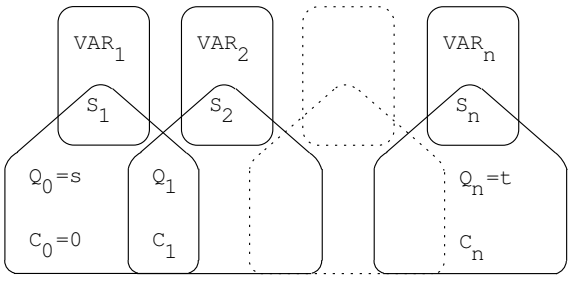


Figure 4.372: Hypergraph of the reformulation corresponding to the automaton of the `sliding_card_skip0` constraint

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 sliding_distribution(SEQ, VARIABLES, VALUES)

Argument(s)

SEQ	:	int
VARIABLES	:	collection(var − dvar)
VALUES	:	collection(val − int, omin − int, omax − int)

Restriction(s)

SEQ >0
SEQ $\leq$ |VARIABLES|
required(VARIABLES, var)
|VALUES| >0
required(VALUES, [val, omin, omax])
distinct(VALUES, val)
VALUES.omin ≥ 0
VALUES.omax $\leq$ SEQ
VALUES.omin $\leq$ VALUES.omax

Purpose For each sequence of SEQ consecutive variables of the VARIABLES collection, each value VALUES$[i]$.val $(1 \leq i \leq |\text{VALUES}|)$ should be taken by at least VALUES$[i]$.omin and at most VALUES$[i]$.omax variables.

Arc input(s) VARIABLES

Arc generator $PATH \mapsto$ collection

Arc arity SEQ

Arc constraint(s) global_cardinality_low_up(collection, VALUES)

Graph property(ies) $\mathbf{NARC} = |\text{VARIABLES}| - \text{SEQ} + 1$

Example

$$\text{sliding\_distribution}\left(\begin{array}{l} 4, \left\{\begin{array}{l} \text{var}-0,\\ \text{var}-5,\\ \text{var}-6,\\ \text{var}-6,\\ \text{var}-5,\\ \text{var}-0,\\ \text{var}-0 \end{array}\right\},\\ \left\{\begin{array}{lll} \text{val}-0 & \text{omin}-1 & \text{omax}-2,\\ \text{val}-1 & \text{omin}-0 & \text{omax}-4,\\ \text{val}-4 & \text{omin}-0 & \text{omax}-4,\\ \text{val}-5 & \text{omin}-1 & \text{omax}-2,\\ \text{val}-6 & \text{omin}-0 & \text{omax}-2 \end{array}\right\} \end{array}\right)$$

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 sliding_sum(LOW, UP, SEQ, VARIABLES)

Argument(s)

LOW	:	int
UP	:	int
SEQ	:	int
VARIABLES	:	collection(var − dvar)

Restriction(s)
- UP ≥ LOW
- SEQ > 0
- SEQ ≤ |VARIABLES|
- required(VARIABLES, var)

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 $PATH \mapsto$ collection

Arc arity SEQ

Arc constraint(s)
- sum_ctr(collection, ≥, LOW)
- sum_ctr(collection, ≤, UP)

Graph property(ies) $\mathbf{NARC} = |\text{VARIABLES}| - \text{SEQ} + 1$

Example

$$\text{sliding\_sum}\left(\left(3, 7, 4, \left\{\begin{array}{l}\text{var}-1,\\ \text{var}-4,\\ \text{var}-2,\\ \text{var}-0,\\ \text{var}-0,\\ \text{var}-3,\\ \text{var}-4\end{array}\right\}\right)\right)$$

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.

Graph model We use sum_ctr as an arc constraint. sum_ctr takes a collection of domain variables as its first argument.

Signature Since we use the *PATH* arc generator with an arity of SEQ on the items of the VARIABLES collection, the expression |VARIABLES| − SEQ + 1 corresponds to the maximum number of arcs of the final graph. Therefore we can rewrite the graph property $\mathbf{NARC} =$ |VARIABLES| − SEQ + 1 to $\mathbf{NARC} \geq$ |VARIABLES| − SEQ + 1 and simplify $\overline{\mathbf{NARC}}$ to $\overline{\mathbf{NARC}}$.

Algorithm [65].

Key words decomposition, sliding sequence constraint, sequence, hypergraph, sum.

4.188 sliding_time_window

Origin N. Beldiceanu

Constraint sliding_time_window(WINDOW_SIZE, LIMIT, TASKS)

Argument(s)

WINDOW_SIZE	:	int
LIMIT	:	int
TASKS	:	collection(id − int, origin − dvar, duration − dvar)

Restriction(s)

- WINDOW_SIZE > 0
- LIMIT ≥ 0
- required(TASKS, [id, origin, duration])
- distinct(TASKS, id)
- TASKS.duration ≥ 0

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 $CLIQUE \mapsto$ collection(tasks1, tasks2)

Arc arity 2

Arc constraint(s)

- tasks1.origin $\leq$ tasks2.origin
- tasks2.origin − tasks1.origin $<$ WINDOW_SIZE

Sets SUCC $\mapsto$ [source, tasks]

Constraint(s) on sets sliding_time_window_from_start(WINDOW_SIZE, LIMIT, tasks, source.origin)

Example

$$\texttt{sliding\_time\_window}\left(\begin{array}{l} 9, 6, \left\{\begin{array}{lll} \texttt{id}-1 & \texttt{origin}-10 & \texttt{duration}-3, \\ \texttt{id}-2 & \texttt{origin}-5 & \texttt{duration}-1, \\ \texttt{id}-3 & \texttt{origin}-6 & \texttt{duration}-2, \\ \texttt{id}-4 & \texttt{origin}-14 & \texttt{duration}-2, \\ \texttt{id}-5 & \texttt{origin}-2 & \texttt{duration}-2 \end{array}\right\} \end{array}\right)$$

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.

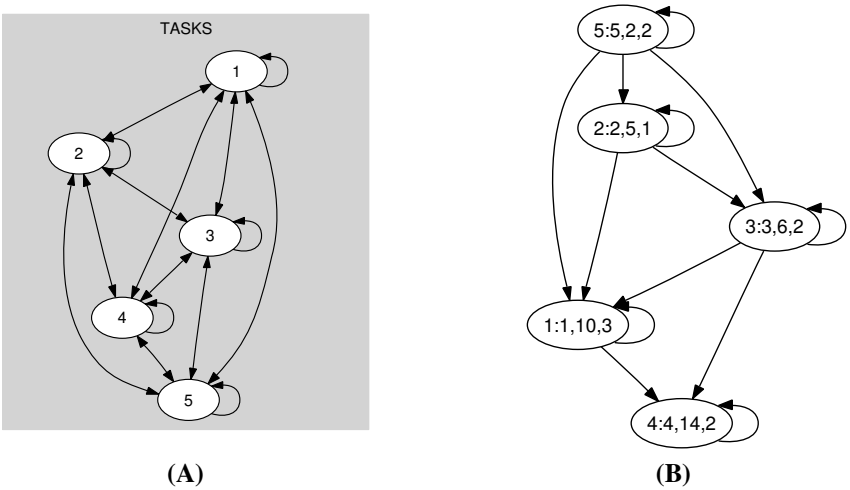


Figure 4.373: Initial and final graph of the `sliding_time_window` constraint

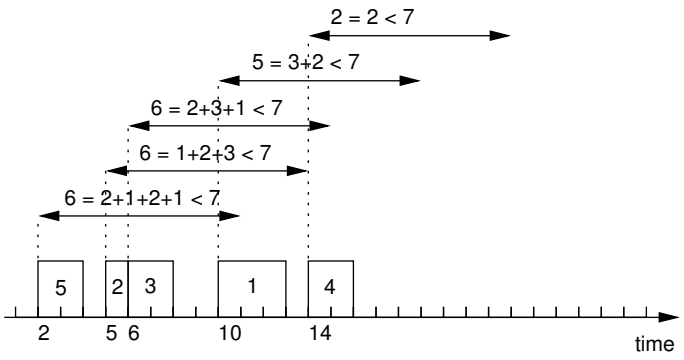


Figure 4.374: Time windows of the `sliding_time_window` constraint

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 `sliding_time_window` constraint is useful for timetabling problems in order to put an upper limit on the total work over sliding time windows.

See also `shift`, `sliding_time_window_from_start`, `sliding_time_window_sum`.

Key words sliding sequence constraint, temporal constraint.

4.189 sliding_time_window_from_start

Origin Used for defining sliding_time_window.

Constraint sliding_time_window_from_start(WINDOW_SIZE, LIMIT, TASKS, START)

Argument(s)

WINDOW_SIZE	:	int
LIMIT	:	int
TASKS	:	collection(id − int, origin − dvar, duration − dvar)
START	:	dvar

Restriction(s)

WINDOW_SIZE > 0
LIMIT ≥ 0
required(TASKS, [id, origin, duration])
distinct(TASKS, id)
TASKS.duration ≥ 0

Purpose The sum of the intersections of all the tasks of the TASKS collection with interval [START, START + WINDOW_SIZE − 1] is less than or equal to LIMIT.

Derived Collection(s) col(S − collection(var − dvar), [item(var − START)])

Arc input(s) S TASKS

Arc generator $PRODUCT \mapsto$ collection(s, tasks)

Arc arity 2

Arc constraint(s) TRUE

Graph property(ies) $$\mathbf{SUM\_WEIGHT\_ARC}\left(\ \texttt{max}\left(\ 0,\ \begin{array}{l}\texttt{min}(\texttt{s.var} + \texttt{WINDOW\_SIZE}, \texttt{tasks.origin} + \texttt{tasks.duration}) - \\ \texttt{max}(\texttt{s.var}, \texttt{tasks.origin})\end{array}\ \right)\ \right)$$

Example $$\texttt{sliding\_time\_window}\left(\ 9, 6, \left\{\begin{array}{lll}\texttt{id} - 1 & \texttt{origin} - 10 & \texttt{duration} - 3, \\ \texttt{id} - 2 & \texttt{origin} - 5 & \texttt{duration} - 1, \\ \texttt{id} - 3 & \texttt{origin} - 6 & \texttt{duration} - 2\end{array}\right\}, 5\ \right)$$

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 [START, START + WINDOW_SIZE − 1]. The constraint sliding_time_window_from_start holds since the sum of the previous intersections does not exceed LIMIT.

Graph model Since we use the TRUE arc constraint the final and the initial graph are identical. The unique source of the final graph corresponds to the interval [START, START + WINDOW_SIZE − 1]. Each sink of the final graph represents a given task of the TASKS collection. We valuate each arc by the intersection of the task associated to one of the extremities of the arc with the time window [START, START + WINDOW_SIZE − 1]. Finally, the graph property **SUM_WEIGHT_ARC** sums up all the valuations of the arcs and check that it does not exceed a given limit.

Used in `sliding_time_window`.

Key words sliding sequence constraint, temporal constraint, derived collection.

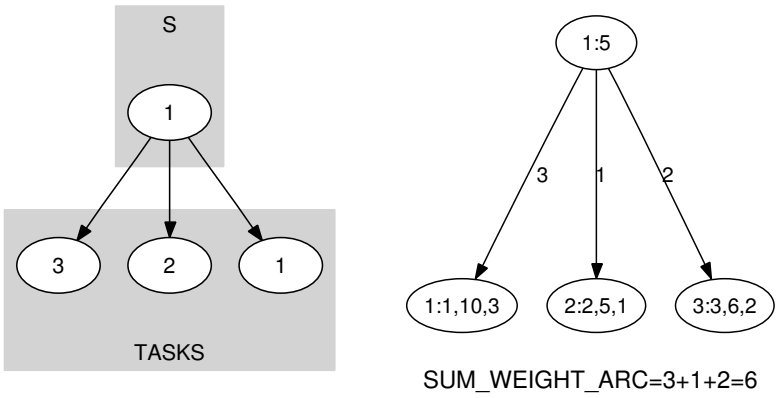


Figure 4.375: Initial and final graph of the `sliding_time_window_from_start` constraint

4.190 sliding_time_window_sum

Origin Derived from sliding_time_window.

Constraint sliding_time_window_sum(WINDOW_SIZE, LIMIT, TASKS)

Argument(s)

WINDOW_SIZE	:	int
LIMIT	:	int
TASKS	:	collection(id − int, origin − dvar, end − dvar, npoint − dvar)

Restriction(s)
- WINDOW_SIZE > 0
- LIMIT ≥ 0
- required(TASKS, [id, origin, end, npoint])
- distinct(TASKS, id)
- TASKS.npoint ≥ 0

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) TASKS

Arc generator $SELF \mapsto$ collection(tasks)

Arc arity 1

Arc constraint(s) tasks.origin $\leq$ tasks.end

Graph property(ies) $\mathbf{NARC} = |$TASKS$|$

Arc input(s) TASKS

Arc generator $CLIQUE \mapsto$ collection(tasks1, tasks2)

Arc arity 2

Arc constraint(s)
- tasks1.end $\leq$ tasks2.end
- tasks2.origin − tasks1.end $<$ WINDOW_SIZE − 1

Sets

$$\text{SUCC} \mapsto \left[\begin{array}{l} \text{source}, \\ \text{variables} - \text{col}\left(\begin{array}{l} \text{VARIABLES} - \text{collection}(\text{var} - \text{dvar}), \\ [\text{item}(\text{var} - \text{TASKS.npoint})] \end{array}\right) \end{array}\right]$$

Constraint(s) on sets sum_ctr(variables, $\leq$, LIMIT)

Example

$$\texttt{sliding\_time\_window\_sum}\left(\begin{array}{l} 9, 16, \left\{\begin{array}{llll} \texttt{id}-1 & \texttt{origin}-10 & \texttt{end}-13 & \texttt{npoint}-2, \\ \texttt{id}-2 & \texttt{origin}-5 & \texttt{end}-6 & \texttt{npoint}-3, \\ \texttt{id}-3 & \texttt{origin}-6 & \texttt{end}-8 & \texttt{npoint}-4, \\ \texttt{id}-4 & \texttt{origin}-14 & \texttt{end}-16 & \texttt{npoint}-5, \\ \texttt{id}-5 & \texttt{origin}-2 & \texttt{end}-4 & \texttt{npoint}-6 \end{array}\right\} \end{array}\right)$$

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.

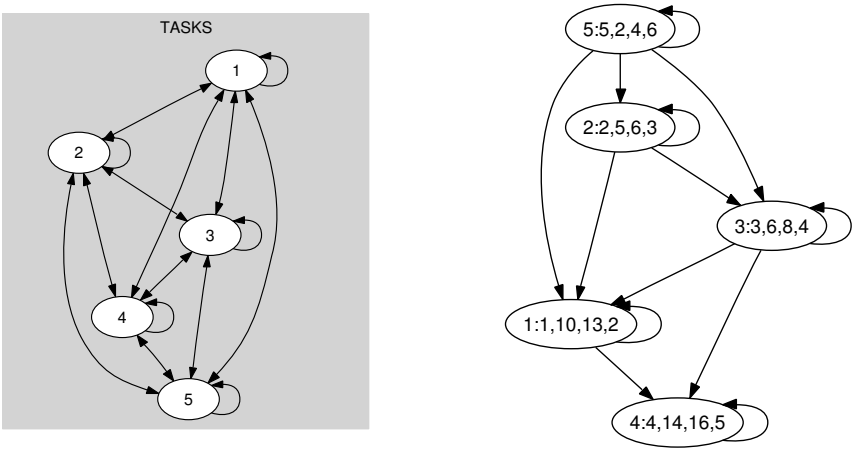


Figure 4.376: Initial and final graph of the `sliding_time_window_sum` constraint

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 |`TASKS`|. Therefore we can rewrite $\mathbf{NARC} =$ |`TASKS`| to $\mathbf{NARC} \geq$ |`TASKS`| and simplify $\underline{\overline{\mathbf{NARC}}}$ to $\overline{\mathbf{NARC}}$.

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.

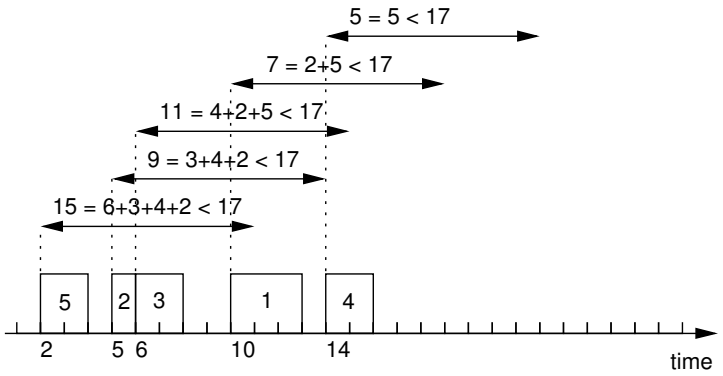


Figure 4.377: Time windows of the `sliding_time_window_sum` constraint

4.191 smooth

Origin Derived from change.

Constraint smooth(NCHANGE, TOLERANCE, VARIABLES)

Argument(s)

NCHANGE	:	dvar
TOLERANCE	:	int
VARIABLES	:	collection(var − dvar)

Restriction(s)

NCHANGE ≥ 0
NCHANGE $<$ |VARIABLES|
TOLERANCE ≥ 0
required(VARIABLES, var)

Purpose NCHANGE is the number of times that $|X - Y| >$ TOLERANCE holds; X and Y correspond to consecutive variables of the collection VARIABLES.

Arc input(s) VARIABLES

Arc generator $PATH \mapsto$ collection(variables1, variables2)

Arc arity 2

Arc constraint(s) abs(variables1.var − variables2.var) $>$ TOLERANCE

Graph property(ies) **NARC** = NCHANGE

Example

$$\texttt{smooth}\left(\begin{array}{c} 1, 2, \left\{\begin{array}{l} \texttt{var} - 1, \\ \texttt{var} - 3, \\ \texttt{var} - 4, \\ \texttt{var} - 5, \\ \texttt{var} - 2 \end{array}\right\} \end{array}\right)$$

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.

Automaton Figure 4.379 depicts the automaton associated to the smooth constraint. To each pair of consecutive variables $(\texttt{VAR}_i, \texttt{VAR}_{i+1})$ of the collection VARIABLES corresponds a 0-1 signature variable $\texttt{S}_i$. The following signature constraint links $\texttt{VAR}_i$, $\texttt{VAR}_{i+1}$ and $\texttt{S}_i$: $(|\texttt{VAR}_i - \texttt{VAR}_{i+1}|) >$ TOLERANCE $\Leftrightarrow \texttt{S}_i = 1$.

Usage This constraint is useful for the following problems:

- 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 smooth constraint you can state a limit on the number of drastic changes.

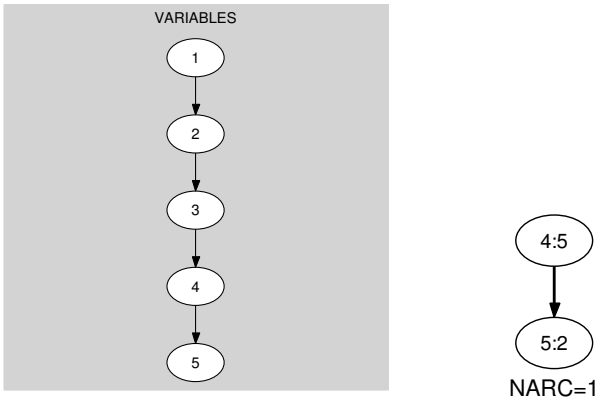


Figure 4.378: Initial and final graph of the smooth constraint

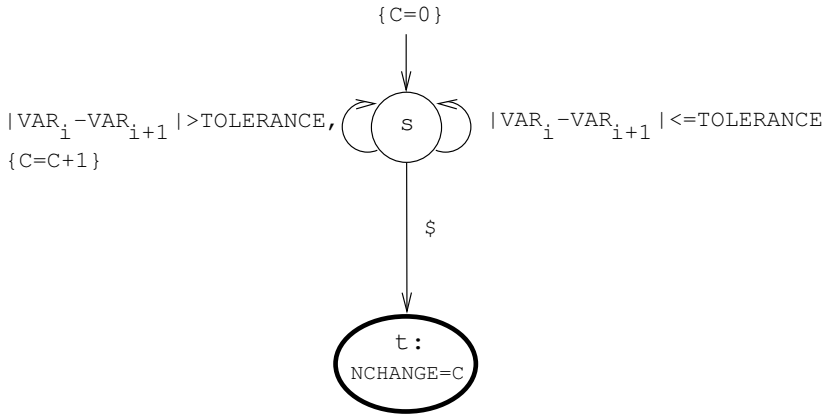


Figure 4.379: Automaton of the smooth constraint

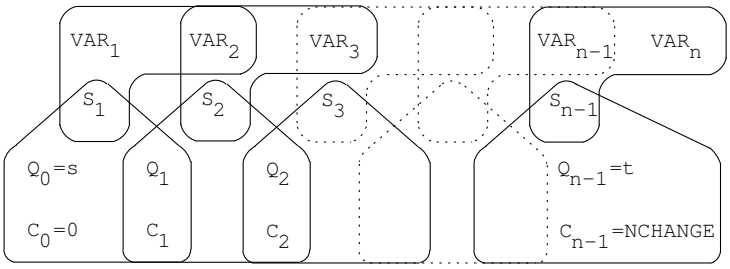


Figure 4.380: Hypergraph of the reformulation corresponding to the automaton of the smooth constraint

- 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 consecutives 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).

4.192 soft_alldifferent_ctr

Origin [10]

Constraint soft_alldifferent_ctr(C, VARIABLES)

Synonym(s) soft_alldiff_ctr, soft_alldistinct_ctr.

Argument(s)

C	:	dvar
VARIABLES	:	collection(var − dvar)

Restriction(s)

C ≥ 0
C ≤ (|VARIABLES| ∗ |VARIABLES| − |VARIABLES|)/2
required(VARIABLES, var)

Purpose Consider the *disequality* constraints involving two distinct variables of the collection VARIABLES. Among the previous set of constraints, C is the number of *disequality* constraints which do not hold.

Arc input(s) VARIABLES

Arc generator $CLIQUE(<) \mapsto$ collection(variables1, variables2)

Arc arity 2

Arc constraint(s) variables1.var = variables2.var

Graph property(ies) $\mathbf{NARC} = \mathtt{C}$

Example $\mathtt{soft\_alldifferent\_ctr}\left(\begin{array}{l} 4, \left\{\begin{array}{l} \mathtt{var}-5,\\ \mathtt{var}-1,\\ \mathtt{var}-9,\\ \mathtt{var}-1,\\ \mathtt{var}-5,\\ \mathtt{var}-5 \end{array}\right\} \end{array}\right)$

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.

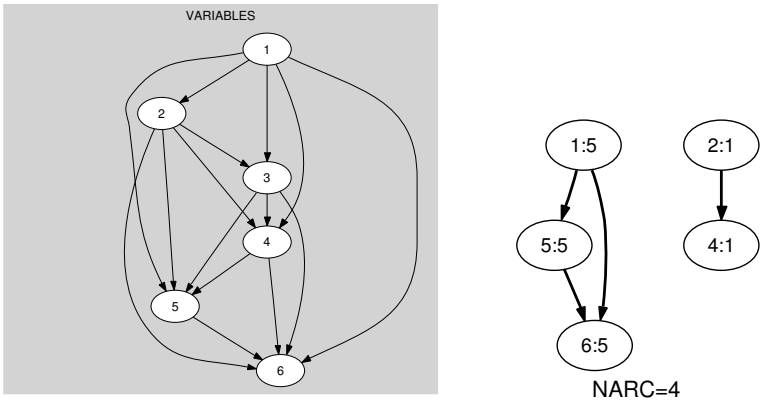


Figure 4.381: Initial and final graph of the `soft_alldifferent_ctr` constraint

4.193 soft_alldifferent_var

Origin [10]

Constraint soft_alldifferent_var(C, VARIABLES)

Synonym(s) soft_alldiff_var, soft_alldistinct_var.

Argument(s)

C	:	dvar
VARIABLES	:	collection(var − dvar)

Restriction(s)

C ≥ 0
C < |VARIABLES|
required(VARIABLES, var)

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.

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** = |VARIABLES| − C

Example soft_alldifferent_var$\left(\begin{array}{c} 3, \left\{\begin{array}{l} \mathtt{var}-5, \\ \mathtt{var}-1, \\ \mathtt{var}-9, \\ \mathtt{var}-1, \\ \mathtt{var}-5, \\ \mathtt{var}-5 \end{array}\right\} \end{array}\right)$

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 *cost* variable C is equal to $6 - 3$.

Graph model We generate a clique with binary *equalities* 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 alldifferent constraint.

Remark Since it focus on the soft aspect of the alldifferent 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.

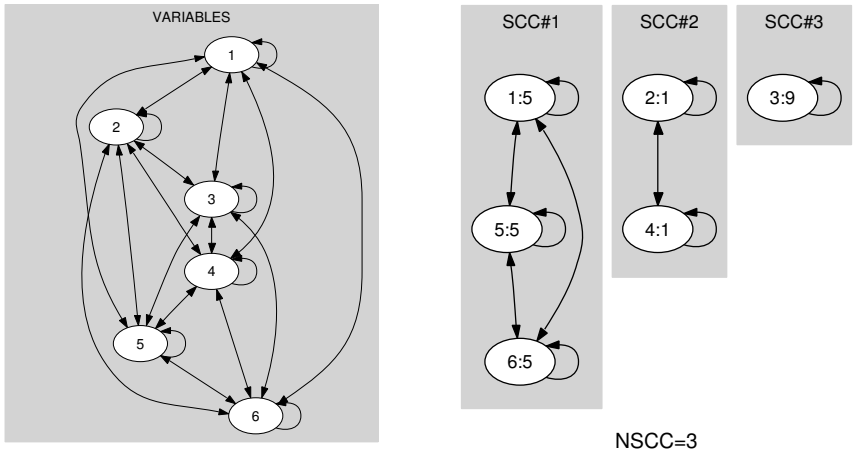


Figure 4.382: Initial and final graph of the `soft_alldifferent_var` constraint

4.194 soft_same_interval_var

Origin Derived from same_interval

Constraint soft_same_interval_var(C, VARIABLES1, VARIABLES2, SIZE_INTERVAL)

Synonym(s) soft_same_interval.

Argument(s)

C	:	dvar
VARIABLES1	:	collection(var − dvar)
VARIABLES2	:	collection(var − dvar)
SIZE_INTERVAL	:	int

Restriction(s)

C ≥ 0
C ≤ |VARIABLES1|
|VARIABLES1| = |VARIABLES2|
required(VARIABLES1, var)
required(VARIABLES2, var)
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 $[\texttt{SIZE\_INTERVAL} \cdot i, \texttt{SIZE\_INTERVAL} \cdot i + \texttt{SIZE\_INTERVAL} - 1$. C is the minimum number of values to change in the VARIABLES1 and VARIABLES2 collections so that for all integer i we have $N_i = M_i$.

Arc input(s) VARIABLES1 VARIABLES2

Arc generator $PRODUCT \mapsto$ collection(variables1, variables2)

Arc arity 2

Arc constraint(s) variables1.var/SIZE_INTERVAL = variables2.var/SIZE_INTERVAL

Graph property(ies) **NSINK_NSOURCE** = |VARIABLES1| − C

Example

$$\texttt{soft\_same\_interval\_var}\left(\begin{array}{l} 4, \left\{\begin{array}{l}\texttt{var}-9,\\ \texttt{var}-9,\\ \texttt{var}-9,\\ \texttt{var}-9,\\ \texttt{var}-9,\\ \texttt{var}-1\end{array}\right\},\\ \left\{\begin{array}{l}\texttt{var}-9,\\ \texttt{var}-1,\\ \texttt{var}-1,\\ \texttt{var}-1,\\ \texttt{var}-1,\\ \texttt{var}-8\end{array}\right\}, 3 \end{array}\right)$$

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.

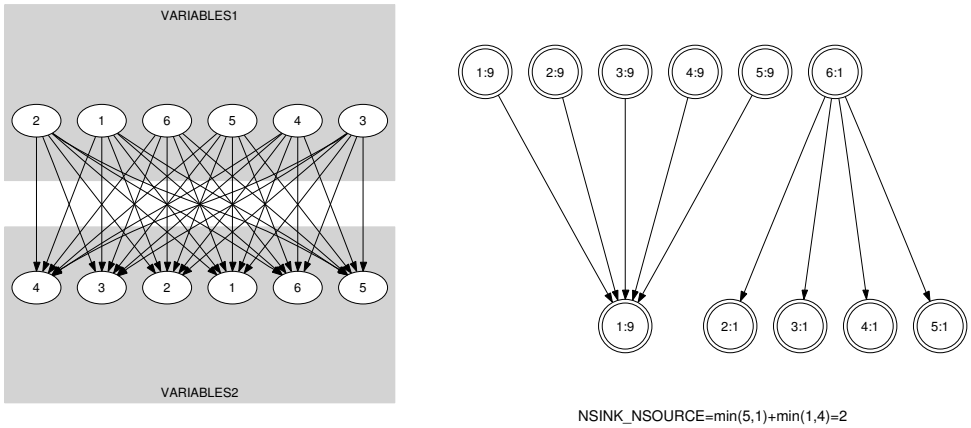


Figure 4.383: Initial and final graph of the `soft_same_interval_var` constraint

Usage A soft `same_interval` constraint.

Algorithm See algorithm of the `soft_same_var` constraint.

See also `same_interval`.

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)

$\texttt{C} \geq 0$
$\texttt{C} \leq |\texttt{VARIABLES1}|$
$|\texttt{VARIABLES1}| = |\texttt{VARIABLES2}|$
required(VARIABLES1, var)
required(VARIABLES2, var)
$\texttt{M} > 0$

Purpose For each integer R in $[0, \texttt{M} - 1]$, 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 $[0, \texttt{M} - 1]$ we have $N1_R = N2_R$.

Arc input(s) VARIABLES1 VARIABLES2

Arc generator $PRODUCT \mapsto$ collection(variables1, variables2)

Arc arity 2

Arc constraint(s) $\texttt{variables1.var} \bmod \texttt{M} = \texttt{variables2.var} \bmod \texttt{M}$

Graph property(ies) $\mathbf{NSINK\_NSOURCE} = |\texttt{VARIABLES1}| - \texttt{C}$

Example

$$\texttt{soft\_same\_modulo\_var}\left(\begin{array}{l} 4, \left\{\begin{array}{l}\texttt{var}-9,\\ \texttt{var}-9,\\ \texttt{var}-9,\\ \texttt{var}-9,\\ \texttt{var}-9,\\ \texttt{var}-1\end{array}\right\},\\ \left\{\begin{array}{l}\texttt{var}-9,\\ \texttt{var}-1,\\ \texttt{var}-1,\\ \texttt{var}-1,\\ \texttt{var}-1,\\ \texttt{var}-8\end{array}\right\}, 3 \end{array}\right)$$

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.

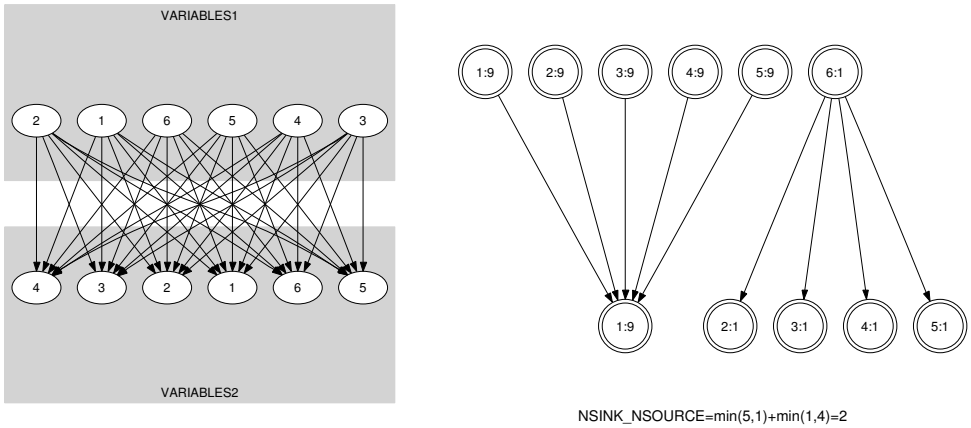


Figure 4.384: Initial and final graph of the `soft_same_modulo_var` constraint

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 same_partition

Constraint soft_same_partition_var(C, VARIABLES1, VARIABLES2, PARTITIONS)

Synonym(s) soft_same_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 ≥ 0
C ≤ |VARIABLES1|
|VARIABLES1| = |VARIABLES2|
required(VARIABLES1, var)
required(VARIABLES2, var)
required(PARTITIONS, p)
|PARTITIONS| ≥ 2
required(VALUES, val)
distinct(VALUES, val)

Purpose

For each integer i in $[1, |\texttt{PARTITIONS}|]$, 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 $[1, |\texttt{PARTITIONS}|]$ we have $N1_i = N2_i$.

Arc input(s) VARIABLES1 VARIABLES2

Arc generator $PRODUCT \mapsto$ collection(variables1, variables2)

Arc arity 2

Arc constraint(s) in_same_partition(variables1.var, variables2.var, PARTITIONS)

Graph property(ies) **NSINK_NSOURCE** = |VARIABLES1| − C

Example

$$\texttt{soft\_same\_partition\_var}\left(\begin{array}{l} 4, \left\{\begin{array}{l}\texttt{var}-9,\\ \texttt{var}-9,\\ \texttt{var}-9,\\ \texttt{var}-9,\\ \texttt{var}-9,\\ \texttt{var}-1\end{array}\right\},\\ \left\{\begin{array}{l}\texttt{var}-9,\\ \texttt{var}-1,\\ \texttt{var}-1,\\ \texttt{var}-1,\\ \texttt{var}-1,\\ \texttt{var}-8\end{array}\right\},\\ \left\{\begin{array}{l}\texttt{p}-\{\texttt{val}-1,\texttt{val}-2\},\\ \texttt{p}-\{\texttt{val}-9\},\\ \texttt{p}-\{\texttt{val}-7,\texttt{val}-8\}\end{array}\right\}\end{array}\right)$$

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.

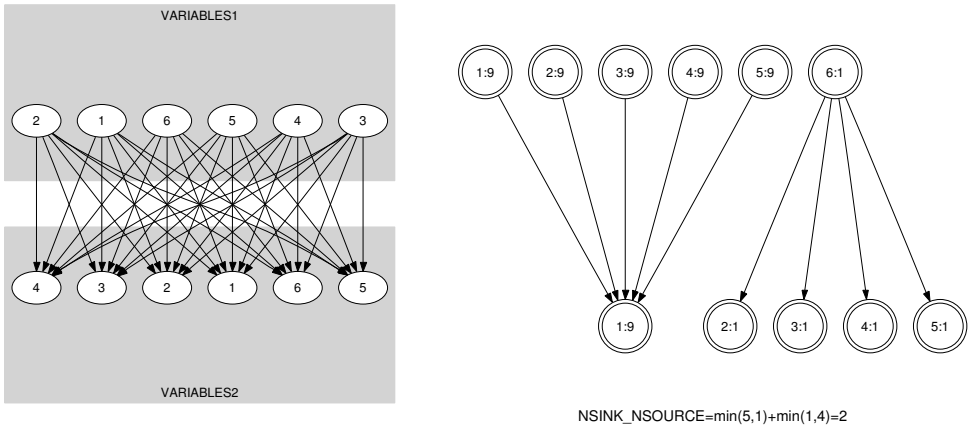


Figure 4.385: Initial and final graph of the `soft_same_partition_var` constraint

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 soft_same_var(C, VARIABLES1, VARIABLES2)

Synonym(s) soft_same.

Argument(s)

C	:	dvar
VARIABLES1	:	collection(var − dvar)
VARIABLES2	:	collection(var − dvar)

Restriction(s)

C ≥ 0
C ≤ |VARIABLES1|
|VARIABLES1| = |VARIABLES2|
required(VARIABLES1, var)
required(VARIABLES2, var)

Purpose C is the minimum number of values to change in the VARIABLES1 and VARIABLES2 collections so that 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 $PRODUCT \mapsto$ collection(variables1, variables2)

Arc arity 2

Arc constraint(s) variables1.var = variables2.var

Graph property(ies) **NSINK_NSOURCE** = |VARIABLES1| − C

Example

$$\texttt{soft\_same\_var}\left(\begin{array}{l} 4, \left\{\begin{array}{l}\texttt{var}-9,\\ \texttt{var}-9,\\ \texttt{var}-9,\\ \texttt{var}-9,\\ \texttt{var}-9,\\ \texttt{var}-1\end{array}\right\},\\ \left\{\begin{array}{l}\texttt{var}-9,\\ \texttt{var}-1,\\ \texttt{var}-1,\\ \texttt{var}-1,\\ \texttt{var}-1,\\ \texttt{var}-8\end{array}\right\}\end{array}\right)$$

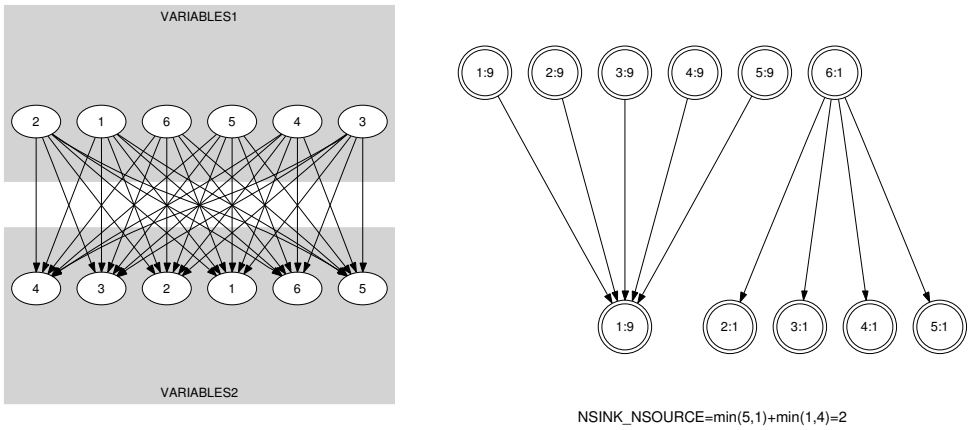
Parts (A) and (B) of Figure 4.386 respectively show the initial and final graph. Since we use the **NSINK_NSOURCE** graph property, the source and sink vertices of the final graph are stressed with a double circle. The soft_same_var constraint holds since the cost 4 corresponds to the difference between 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.



(A) **(B)**

Figure 4.386: Initial and final graph of the `soft_same_var` constraint

4.198 soft_used_by_interval_var

Origin Derived from used_by_interval.

Constraint soft_used_by_interval_var(C, VARIABLES1, VARIABLES2, SIZE_INTERVAL)

Synonym(s) soft_used_by_interval.

Argument(s)

C	:	dvar
VARIABLES1	:	collection(var − dvar)
VARIABLES2	:	collection(var − dvar)
SIZE_INTERVAL	:	int

Restriction(s)

C ≥ 0
C ≤ |VARIABLES2|
|VARIABLES1| ≥ |VARIABLES2|
required(VARIABLES1, var)
required(VARIABLES2, var)
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 [SIZE_INTERVAL $\cdot\, i$, SIZE_INTERVAL $\cdot\, i +$ 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 $PRODUCT \mapsto$ collection(variables1, variables2)

Arc arity 2

Arc constraint(s) variables1.var/SIZE_INTERVAL = variables2.var/SIZE_INTERVAL

Graph property(ies) **NSINK_NSOURCE** = |VARIABLES2| − C

Example

$$\text{soft\_used\_by\_interval\_var}\left(\begin{array}{l} 2, \left\{\begin{array}{l}\text{var}-9,\\ \text{var}-1,\\ \text{var}-1,\\ \text{var}-8,\\ \text{var}-8\end{array}\right\},\\ \left\{\begin{array}{l}\text{var}-9,\\ \text{var}-9,\\ \text{var}-9,\\ \text{var}-1\end{array}\right\}, 3 \end{array}\right)$$

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.

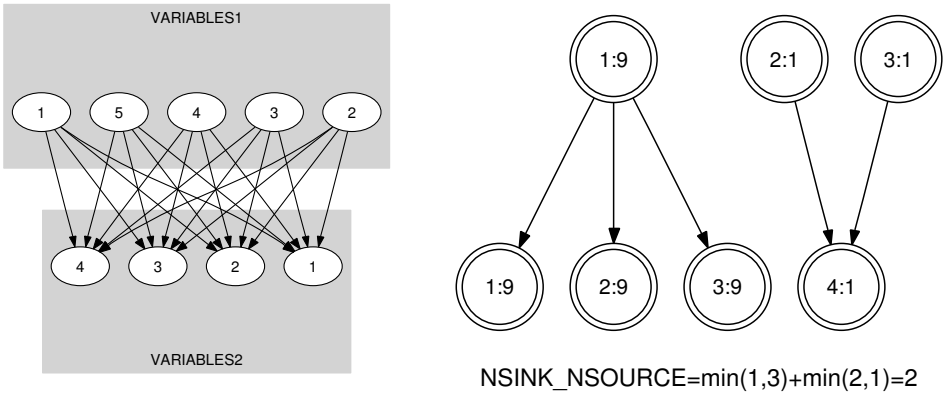


Figure 4.387: Initial and final graph of the `soft_used_by_interval_var` constraint

4.199 soft_used_by_modulo_var

Origin Derived from used_by_modulo

Constraint soft_used_by_modulo_var(C, VARIABLES1, VARIABLES2, M)

Synonym(s) soft_used_by_modulo.

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) VARIABLES1 VARIABLES2

Arc generator $PRODUCT \mapsto$ collection(variables1, variables2)

Arc arity 2

Arc constraint(s) variables1.var mod M = variables2.var mod M

Graph property(ies) **NSINK_NSOURCE** = |VARIABLES2| − C

Example

$$\texttt{soft\_used\_by\_modulo\_var}\left(\begin{array}{l} 2, \left\{\begin{array}{l}\texttt{var}-9,\\ \texttt{var}-1,\\ \texttt{var}-1,\\ \texttt{var}-8,\\ \texttt{var}-8\end{array}\right\},\\ \left\{\begin{array}{l}\texttt{var}-9,\\ \texttt{var}-9,\\ \texttt{var}-9,\\ \texttt{var}-1\end{array}\right\}, 3\end{array}\right)$$

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.

Usage A soft `used_by_modulo` constraint.

Key words soft constraint, constraint between two collections of variables, relaxation, variable-based violation measure, modulo.

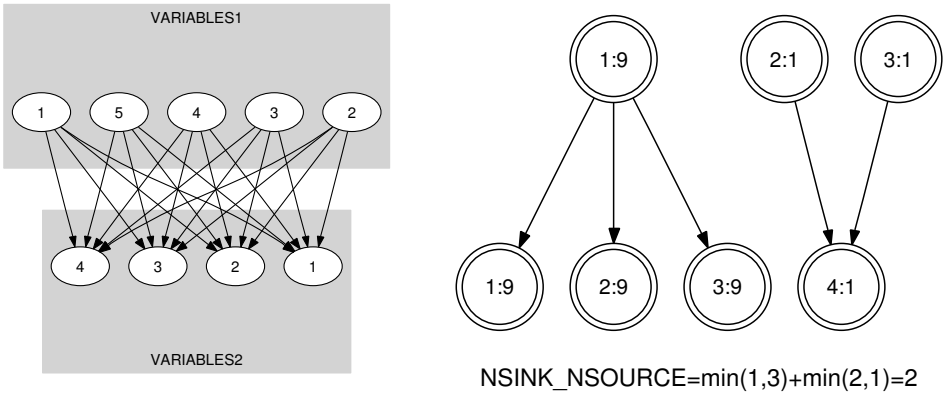


Figure 4.388: Initial and final graph of the `soft_used_by_modulo_var` constraint

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 ≥ 0
C ≤ |VARIABLES2|
|VARIABLES1| ≥ |VARIABLES2|
required(VARIABLES1, var)
required(VARIABLES2, var)
required(PARTITIONS, p)
|PARTITIONS| ≥ 2
required(VALUES, val)
distinct(VALUES, val)

Purpose
For each integer i in $[1, |\texttt{PARTITIONS}|]$, 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 $[1, |\texttt{PARTITIONS}|]$ we have $N1_i = N2_i$.

Arc input(s) VARIABLES1 VARIABLES2

Arc generator $PRODUCT \mapsto$ collection(variables1, variables2)

Arc arity 2

Arc constraint(s) in_same_partition(variables1.var, variables2.var, PARTITIONS)

Graph property(ies) **NSINK_NSOURCE** = |VARIABLES2| − C

Example

$$\texttt{soft\_used\_by\_partition\_var}\left(\begin{array}{l} 2, \left\{\begin{array}{l}\texttt{var}-9,\\ \texttt{var}-1,\\ \texttt{var}-1,\\ \texttt{var}-8,\\ \texttt{var}-8\end{array}\right\},\\ \left\{\begin{array}{l}\texttt{var}-9,\\ \texttt{var}-9,\\ \texttt{var}-9,\\ \texttt{var}-1\end{array}\right\},\\ \left\{\begin{array}{l}\texttt{p}-\{\texttt{val}-1,\texttt{val}-2\},\\ \texttt{p}-\{\texttt{val}-9\},\\ \texttt{p}-\{\texttt{val}-7,\texttt{val}-8\}\end{array}\right\}\end{array}\right)$$

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.

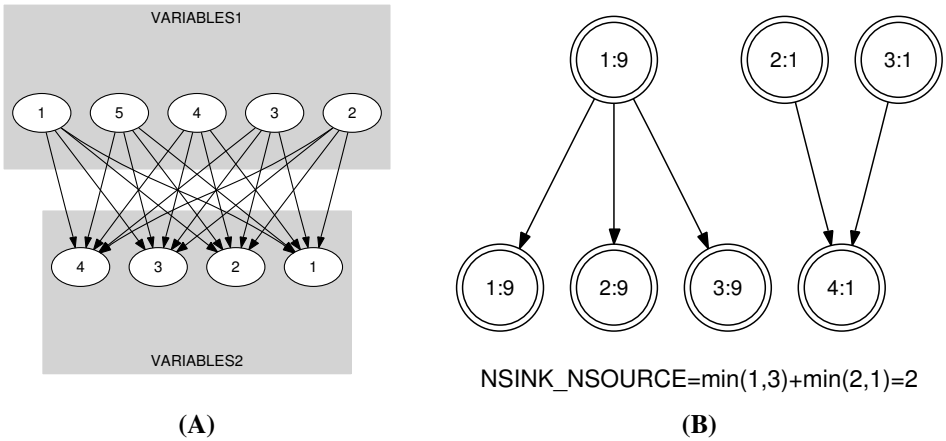


Figure 4.389: Initial and final graph of the `soft_used_by_partition_var` constraint

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 used_by

Constraint soft_used_by_var(C, VARIABLES1, VARIABLES2)

Synonym(s) soft_used_by.

Argument(s)

C	:	dvar
VARIABLES1	:	collection(var − dvar)
VARIABLES2	:	collection(var − dvar)

Restriction(s)

$C \geq 0$
$C \leq |\texttt{VARIABLES2}|$
$|\texttt{VARIABLES1}| \geq |\texttt{VARIABLES2}|$
required(VARIABLES1, var)
required(VARIABLES2, var)

Purpose C is the minimum number of values to change in the VARIABLES1 and VARIABLES2 collections so that all the values of the variables of collection VARIABLES2 are used by the variables of collection VARIABLES1.

Arc input(s) VARIABLES1 VARIABLES2

Arc generator $PRODUCT \mapsto$ collection(variables1, variables2)

Arc arity 2

Arc constraint(s) variables1.var = variables2.var

Graph property(ies) **NSINK_NSOURCE** $= |\texttt{VARIABLES2}| - \texttt{C}$

Example

$$\texttt{soft\_used\_by\_var}\left(\begin{array}{l} 2, \left\{\begin{array}{l}\texttt{var}-9,\\ \texttt{var}-1,\\ \texttt{var}-1,\\ \texttt{var}-8,\\ \texttt{var}-8\end{array}\right\},\\ \left\{\begin{array}{l}\texttt{var}-9,\\ \texttt{var}-9,\\ \texttt{var}-9,\\ \texttt{var}-1\end{array}\right\}\end{array}\right)$$

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.

Usage A soft `used_by` constraint.

Key words soft constraint, constraint between two collections of variables, relaxation, variable-based violation measure.

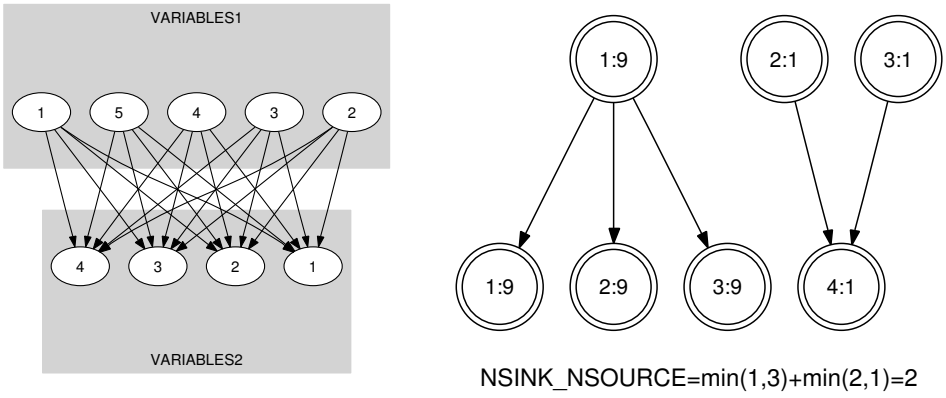


Figure 4.390: Initial and final graph of the `soft_used_by_var` constraint

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 $PRODUCT \mapsto$ collection(variables1, variables2)

Arc arity 2

Arc constraint(s) variables1.var = variables2.var

Graph property(ies)
- for all connected components: **NSOURCE** = **NSINK**
- **NSOURCE** = |VARIABLES1|
- **NSINK** = |VARIABLES2|

Arc input(s) VARIABLES2

Arc generator $PATH \mapsto$ collection(variables1, variables2)

Arc arity 2

Arc constraint(s) variables1.var ≤ variables2.var

Graph property(ies) **NARC** = |VARIABLES2| − 1

Example

$$\texttt{sort}\left(\left\{\begin{array}{l}\texttt{var}-1,\\ \texttt{var}-9,\\ \texttt{var}-1,\\ \texttt{var}-5,\\ \texttt{var}-2,\\ \texttt{var}-1\end{array}\right\},\left\{\begin{array}{l}\texttt{var}-1,\\ \texttt{var}-1,\\ \texttt{var}-1,\\ \texttt{var}-2,\\ \texttt{var}-5,\\ \texttt{var}-9\end{array}\right\}\right)$$

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 |`VARIABLES1`|.
- The number of sinks of the final graph of the first graph constraint is equal to |`VARIABLES2`|.
- Finally the second graph constraint holds also since its corresponding final graph contains exactly |`VARIABLES1` $-$ 1| arcs: All the inequalities constraints between consecutive variables of `VARIABLES2` holds.

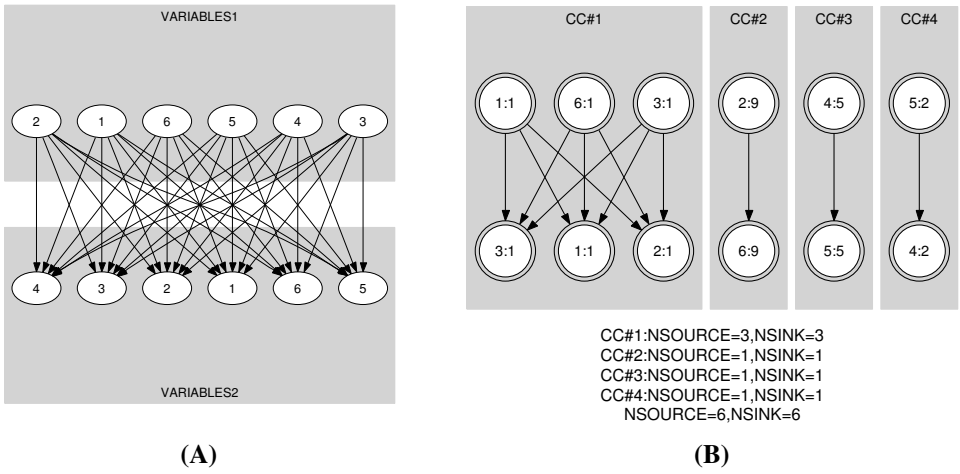


Figure 4.391: Initial and final graph of the `sort` constraint

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 |`VARIABLES1`| and |`VARIABLES2`|. Therefore we can rewrite $\mathbf{NSOURCE} = |\texttt{VARIABLES1}|$ to $\mathbf{NSOURCE} \geq |\texttt{VARIABLES1}|$ and simplify $\underline{\overline{\mathbf{NSOURCE}}}$ to $\overline{\mathbf{NSOURCE}}$. In a similar way, we can rewrite

$\overline{\mathbf{NSINK}} = |\texttt{VARIABLES2}|$ to $\mathbf{NSINK} \geq |\texttt{VARIABLES2}|$ and simplify $\overline{\overline{\mathbf{NSINK}}}$ to $\overline{\mathbf{NSINK}}$.

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 $|\texttt{VARIABLES2}| - 1$. Therefore we can rewrite the graph property $\mathbf{NARC} = |\texttt{VARIABLES2}| - 1$ to $\mathbf{NARC} \geq |\texttt{VARIABLES2}| - 1$ and simplify $\overline{\overline{\mathbf{NARC}}}$ to $\overline{\mathbf{NARC}}$.

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 VARIABLES1 to VARIABLES2.

Algorithm [61, 23].

See also same, sort_permutation.

Key words constraint between two collections of variables, sort, permutation.

4.203 sort_permutation

Origin [147]

Constraint sort_permutation(FROM, PERMUTATION, TO)

Usual name sort

Argument(s)

FROM	:	collection(var − dvar)
PERMUTATION	:	collection(var − dvar)
TO	:	collection(var − dvar)

Restriction(s)
- |PERMUTATION| = |FROM|
- |PERMUTATION| = |TO|
- PERMUTATION.var ≥ 1
- PERMUTATION.var ≤ |PERMUTATION|
- alldifferent(PERMUTATION)
- required(FROM, var)
- required(PERMUTATION, var)
- required(TO, var)

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)

$$\texttt{col}\left(\begin{array}{l}\texttt{FROM\_PERMUTATION} - \texttt{collection}(\texttt{var} - \texttt{dvar}, \texttt{ind} - \texttt{dvar}),\\ [\texttt{item}(\texttt{var} - \texttt{FROM.var}, \texttt{ind} - \texttt{PERMUTATION.var})]\end{array}\right)$$

Arc input(s) FROM_PERMUTATION TO

Arc generator $PRODUCT \mapsto$ collection(from_permutation, to)

Arc arity 2

Arc constraint(s)
- from_permutation.var = to.var
- from_permutation.ind = to.key

Graph property(ies) $\mathbf{NARC} = |\texttt{PERMUTATION}|$

Arc input(s) TO

Arc generator $PATH \mapsto$ collection(to1, to2)

Arc arity 2

Arc constraint(s) to1.var ≤ to2.var

Graph property(ies) $\mathbf{NARC} = |\texttt{TO}| - 1$

Example

$$\texttt{sort\_permutation}\left(\left\{\begin{array}{l}\texttt{var}-1,\\ \texttt{var}-9,\\ \texttt{var}-1,\\ \texttt{var}-5,\\ \texttt{var}-2,\\ \texttt{var}-1\end{array}\right\},\left\{\begin{array}{l}\texttt{var}-1,\\ \texttt{var}-6,\\ \texttt{var}-3,\\ \texttt{var}-5,\\ \texttt{var}-4,\\ \texttt{var}-2\end{array}\right\},\left\{\begin{array}{l}\texttt{var}-1,\\ \texttt{var}-1,\\ \texttt{var}-1,\\ \texttt{var}-2,\\ \texttt{var}-5,\\ \texttt{var}-9\end{array}\right\}\right)$$

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 $|\texttt{PERMUTATION}-1|$ arcs: All the inequalities constraints between consecutive variables of TO holds.

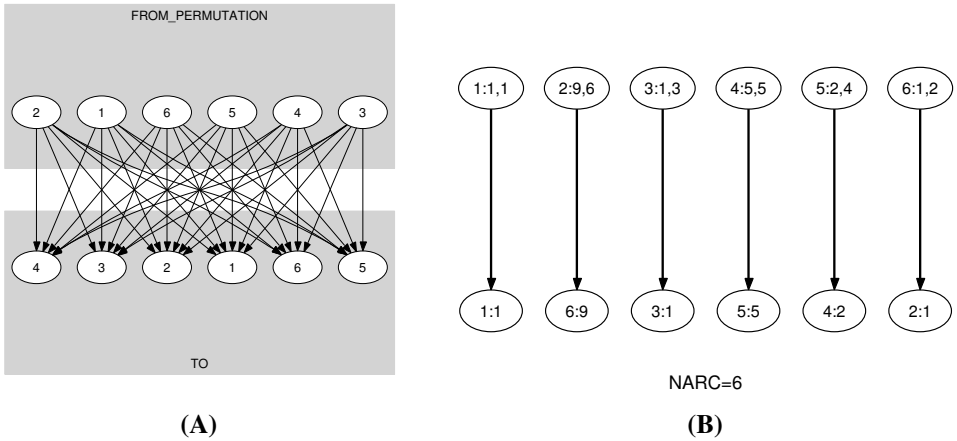


Figure 4.392: Initial and final graph of the sort_permutation constraint

Signature Consider the first graph constraint where we use the $PRODUCT$ arc generator. Since all the `key` attributes of the `TO` 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 |`PERMUTATION`|. So we can rewrite the graph property $\mathbf{NARC} =$ |`PERMUTATION`| to $\mathbf{NARC} \geq$ |`PERMUTATION`| and simplify $\underline{\overline{\mathbf{NARC}}}$ to $\overline{\mathbf{NARC}}$.

Consider now the second graph constraint. Since we use the $PATH$ arc generator with an arity of 2 on the `TO` collection, the maximum number of arcs of the corresponding final graph is equal to |`TO`| $-$ 1. Therefore we can rewrite $\mathbf{NARC} =$ |`TO`| $-$ 1 to $\mathbf{NARC} \geq$ |`TO`| $-$ 1 and simplify $\underline{\overline{\mathbf{NARC}}}$ to $\overline{\mathbf{NARC}}$.

Algorithm [147].

See also `correspondence`, `sort`.

Key words constraint between three collections of variables, sort, permutation, derived collection.

4.204 stage_element

Origin CHOCO, derived from element.

Constraint stage_element(ITEM, TABLE)

Usual name stage_elt

Argument(s)

ITEM	:	collection(index − dvar, value − dvar)
TABLE	:	collection(low − int, up − int, value − int)

Restriction(s)
required(ITEM, [index, value])
$|\texttt{ITEM}| = 1$
required(TABLE, [low, up, value])

Purpose
Let $\texttt{low}_i$, $\texttt{up}_i$ and $\texttt{value}_i$ respectively denote the values of the low, up and value attributes of the i^{th} item of the TABLE collection. First we have that: $\texttt{low}_i \leq \texttt{up}_i$ and $\texttt{up}_i + 1 = \texttt{low}_{i+1}$. Second, the stage_element constraint enforces the following equivalence: $\texttt{low}_i \leq \texttt{ITEM.index} \wedge \texttt{ITEM.index} \leq \texttt{up}_i \Leftrightarrow \texttt{ITEM.value} = \texttt{value}_i$.

Arc input(s) TABLE

Arc generator $PATH \mapsto$ collection(table1, table2)

Arc arity 2

Arc constraint(s)
- table1.low ≤ table1.up
- table1.up + 1 = table2.low
- table2.low ≤ table2.up

Graph property(ies) $\mathbf{NARC} = |\texttt{TABLE}| - 1$

Arc input(s) ITEM TABLE

Arc generator $PRODUCT \mapsto$ collection(item, table)

Arc arity 2

Arc constraint(s)
- item.index ≥ table.low
- item.index ≤ table.up
- item.value = table.value

Graph property(ies) $\mathbf{NARC} = 1$

Example

$$\texttt{stage\_element}\left(\begin{array}{l}\{\texttt{index}-5\ \texttt{value}-6\},\\ \left\{\begin{array}{lll}\texttt{low}-3 & \texttt{up}-7 & \texttt{value}-6,\\ \texttt{low}-8 & \texttt{up}-8 & \texttt{value}-9,\\ \texttt{low}-9 & \texttt{up}-14 & \texttt{value}-2,\\ \texttt{low}-15 & \texttt{up}-19 & \texttt{value}-9\end{array}\right\}\end{array}\right)$$

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 $\texttt{LOW}_i$, $\texttt{UP}_i$ and $\texttt{VALUE}_i$ respectively be the `low`, the `up` and the `value` attributes of the i^{th} item of the `TABLE` collection. To each quintuple $(\texttt{INDEX}, \texttt{VALUE}, \texttt{LOW}_i, \texttt{UP}_i, \texttt{VALUE}_i)$ corresponds a 0-1 signature variable $\texttt{S}_i$ as well as the following signature constraint: $((\texttt{LOW}_i \leq \texttt{INDEX}) \wedge (\texttt{INDEX} \leq \texttt{UP}_i) \wedge (\texttt{VALUE} = \texttt{VALUE}_i)) \Leftrightarrow \texttt{S}_i$.

See also `element`, `elem`.

Key words data constraint, binary constraint, table, functional dependency, automaton, automaton without counters, centered cyclic(2) constraint network(1).

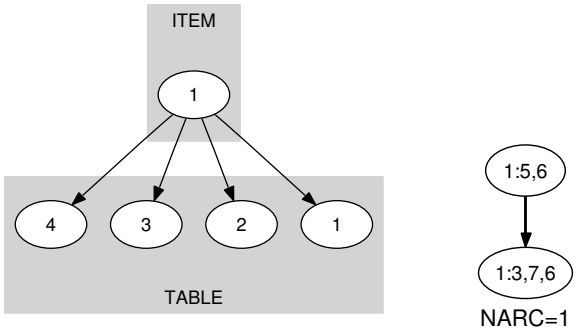


Figure 4.393: Initial and final graph of the `stage_element` constraint

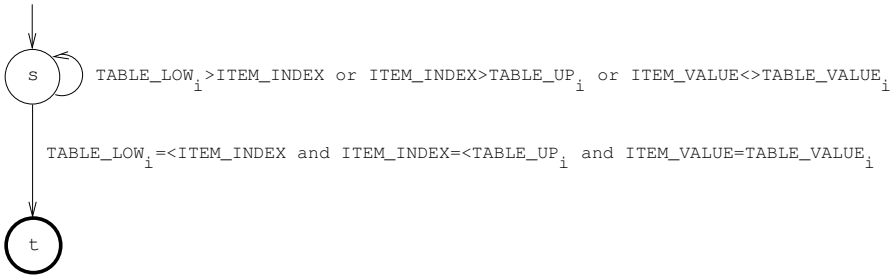


Figure 4.394: Automaton of the `stage_element` constraint

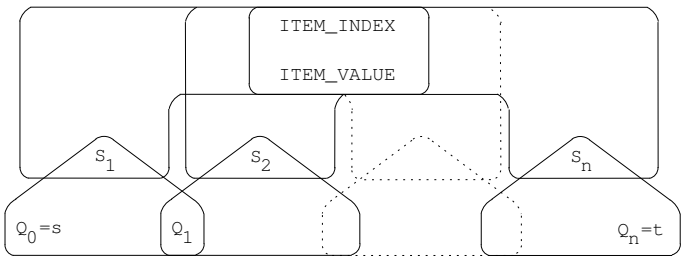


Figure 4.395: Hypergraph of the reformulation corresponding to the automaton of the `stage_element` constraint

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 $X_i, \ldots, X_j$ $(0 \leq i < n, 0 \leq j < n)$ be consecutive variables of the collection of variables VARIABLES such that the following conditions apply:

- All variables $X_i, \ldots, X_j$ 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 $(\texttt{val} - v, \texttt{lmin} - s, \texttt{lmax} - t)$ 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

$CIRCUIT \mapsto$ collection(variables1, variables2)
$LOOP \mapsto$ 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

$$\texttt{stretch\_circuit}\left(\left\{\begin{array}{l}\texttt{var}-6,\\ \texttt{var}-6,\\ \texttt{var}-3,\\ \texttt{var}-1,\\ \texttt{var}-1,\\ \texttt{var}-1,\\ \texttt{var}-6,\\ \texttt{var}-6\end{array}\right\},\left\{\begin{array}{lll}\texttt{val}-1 & \texttt{lmin}-2 & \texttt{lmax}-4,\\ \texttt{val}-2 & \texttt{lmin}-2 & \texttt{lmax}-3,\\ \texttt{val}-3 & \texttt{lmin}-1 & \texttt{lmax}-6,\\ \texttt{val}-6 & \texttt{lmin}-2 & \texttt{lmax}-4\end{array}\right\}\right)$$

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.

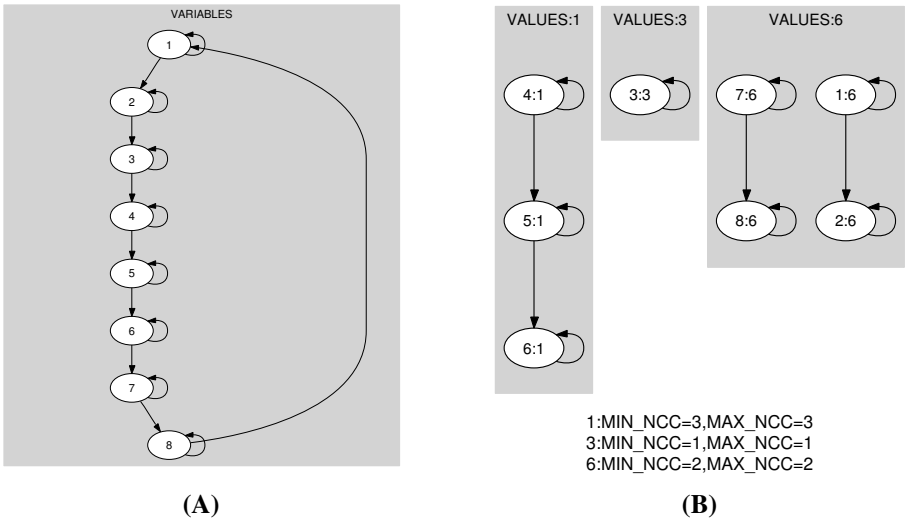


Figure 4.396: Initial and final graph of the stretch_circuit constraint

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.

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 $X_i, \ldots, X_j$ $(1 \leq i \leq j \leq n)$ be consecutive variables of the collection of variables VARIABLES such that the following conditions apply:

- All variables $X_i, \ldots, X_j$ 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
- $PATH \mapsto$ collection(variables1, variables2)
- $LOOP \mapsto$ 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

$$\texttt{stretch\_path}\left(\left\{\begin{array}{l}\texttt{var}-6,\\ \texttt{var}-6,\\ \texttt{var}-3,\\ \texttt{var}-1,\\ \texttt{var}-1,\\ \texttt{var}-1,\\ \texttt{var}-6,\\ \texttt{var}-6\end{array}\right\},\left\{\begin{array}{lll}\texttt{val}-1 & \texttt{lmin}-2 & \texttt{lmax}-4,\\ \texttt{val}-2 & \texttt{lmin}-2 & \texttt{lmax}-3,\\ \texttt{val}-3 & \texttt{lmin}-1 & \texttt{lmax}-6,\\ \texttt{val}-6 & \texttt{lmin}-2 & \texttt{lmax}-2\end{array}\right\}\right)$$

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.

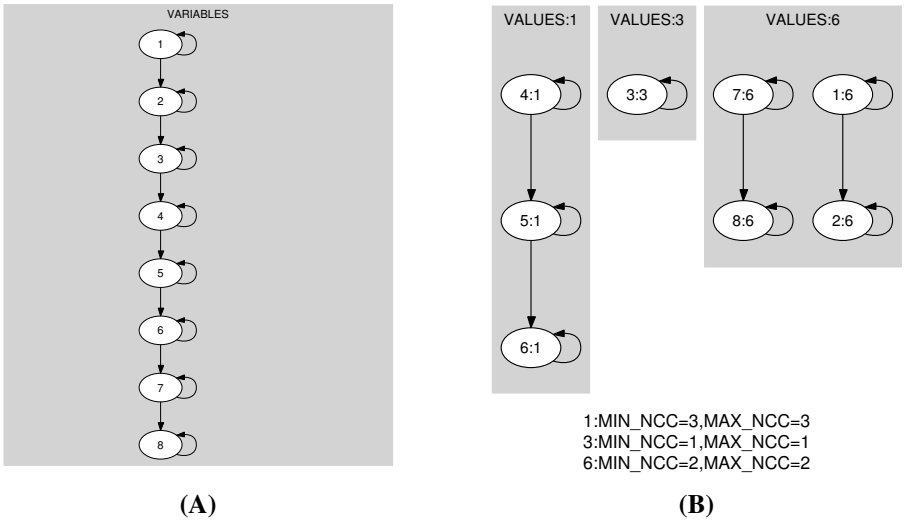


Figure 4.397: Initial and final graph of the stretch_path constraint

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.

4.207 strict_lex2

Origin [123]

Constraint strict_lex2(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 both adjacent rows, and adjacent columns are lexicographically ordered (adjacent rows and adjacent columns cannot be equal).

Example $\texttt{strict\_lex2}\left(\left\{\begin{array}{l}\texttt{vec}-\{\texttt{var}-2,\texttt{var}-2,\texttt{var}-3\},\\ \texttt{vec}-\{\texttt{var}-2,\texttt{var}-3,\texttt{var}-1\}\end{array}\right\}\right)$

Usage A *symmetry-breaking* constraint.

See also lex2, allperm, lex_lesseq, lex_chain_lesseq.

Key words predefined constraint, order constraint, matrix, matrix model, symmetry, lexicographic order.

4.208 strictly_decreasing

Origin Derived from strictly_increasing.

Constraint strictly_decreasing(VARIABLES)

Argument(s) VARIABLES : collection(var − dvar)

Restriction(s) |VARIABLES| > 0
required(VARIABLES, var)

Purpose The variables of the collection VARIABLES are strictly decreasing.

Arc input(s) VARIABLES

Arc generator $PATH \mapsto$ collection(variables1, variables2)

Arc arity 2

Arc constraint(s) variables1.var > variables2.var

Graph property(ies) $\mathbf{NARC} = |\texttt{VARIABLES}| - 1$

Example strictly_decreasing $\left(\left\{ \begin{array}{l} \texttt{var} - 8, \\ \texttt{var} - 4, \\ \texttt{var} - 3, \\ \texttt{var} - 1 \end{array} \right\} \right)$

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.

Automaton Figure 4.399 depicts the automaton associated to the strictly_decreasing constraint. To each pair of consecutive variables $(\texttt{VAR}_i, \texttt{VAR}_{i+1})$ of the collection VARIABLES corresponds a 0-1 signature variable $\texttt{S}_i$. The following signature constraint links $\texttt{VAR}_i$, $\texttt{VAR}_{i+1}$ and $\texttt{S}_i$: $\texttt{VAR}_i \leq \texttt{VAR}_{i+1} \Leftrightarrow \texttt{S}_i$.

See also decreasing, increasing, strictly_increasing.

Key words decomposition, order constraint, automaton, automaton without counters, sliding cyclic(1) constraint network(1).

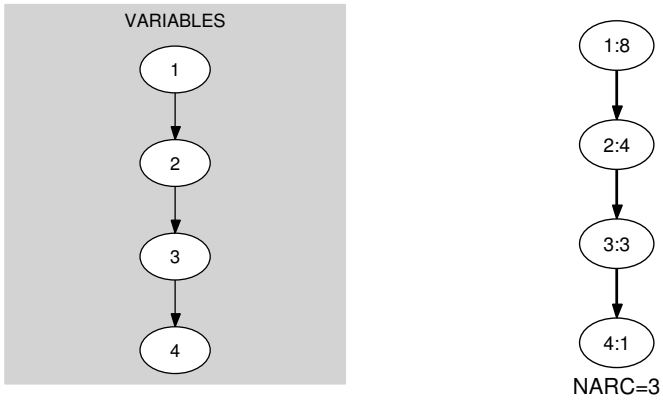


Figure 4.398: Initial and final graph of the `strictly_decreasing` constraint

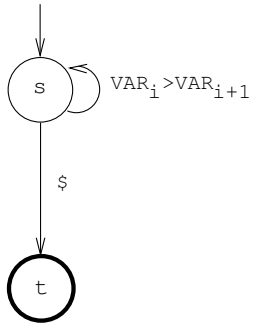


Figure 4.399: Automaton of the `strictly_decreasing` constraint

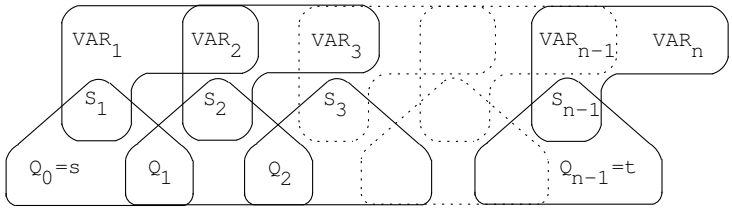


Figure 4.400: Hypergraph of the reformulation corresponding to the automaton of the `strictly_decreasing` constraint

4.209 strictly_increasing

Origin KOALOG

Constraint strictly_increasing(VARIABLES)

Argument(s) VARIABLES : collection(var − dvar)

Restriction(s) |VARIABLES| > 0
required(VARIABLES, var)

Purpose The variables of the collection VARIABLES are strictly increasing.

Arc input(s) VARIABLES

Arc generator $PATH \mapsto$ collection(variables1, variables2)

Arc arity 2

Arc constraint(s) variables1.var < variables2.var

Graph property(ies) $\mathbf{NARC} = |\text{VARIABLES}| - 1$

Example $\text{strictly\_increasing}\left(\left\{\begin{array}{l}\text{var}-1,\\ \text{var}-3,\\ \text{var}-4,\\ \text{var}-8\end{array}\right\}\right)$

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.

Automaton Figure 4.402 depicts the automaton associated to the strictly_increasing 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: $\text{VAR}_i \geq \text{VAR}_{i+1} \Leftrightarrow \text{S}_i$.

See also increasing, decreasing, strictly_decreasing.

Key words decomposition, order constraint, automaton, automaton without counters, sliding cyclic(1) constraint network(1).

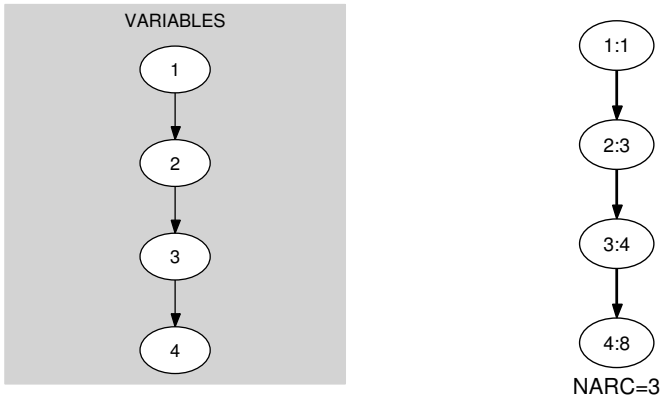


Figure 4.401: Initial and final graph of the `strictly_increasing` constraint

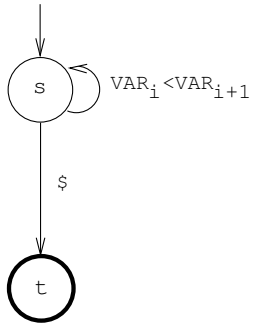


Figure 4.402: Automaton of the `strictly_increasing` constraint

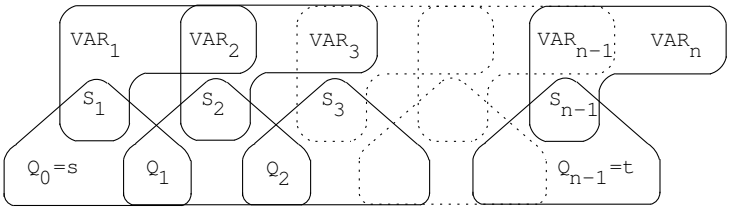


Figure 4.403: Hypergraph of the reformulation corresponding to the automaton of the `strictly_increasing` constraint

4.210 strongly_connected

Origin [74]

Constraint strongly_connected(NODES)

Argument(s) NODES : collection(index − int, succ − svar)

Restriction(s) required(NODES, [index, succ])
NODES.index ≥ 1
NODES.index $\leq$ |NODES|
distinct(NODES, index)

Purpose Consider a digraph G described by the NODES collection. Select a subset of arcs of G so that we have one single strongly connected component involving all 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) $\mathbf{MIN\_NSCC} = |\text{NODES}|$

Example strongly_connected $\left(\left\{ \begin{array}{ll} \text{index} - 1 & \text{succ} - \{2\}, \\ \text{index} - 2 & \text{succ} - \{3\}, \\ \text{index} - 3 & \text{succ} - \{2, 5\}, \\ \text{index} - 4 & \text{succ} - \{1\}, \\ \text{index} - 5 & \text{succ} - \{4\} \end{array} \right\} \right)$

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 succ attribute of a given vertex. Part (B) of Figure 4.404 gives the final graph associated to the example. The strongly_connected constraint holds since the final graph contains one single strongly connected component mentioning every vertex of the initial graph.

Signature Since the maximum number of vertices of the final graph is equal to |NODES| we can rewrite the graph property $\mathbf{MIN\_NSCC} = |\text{NODES}|$ to $\mathbf{MIN\_NSCC} \geq |\text{NODES}|$ and simplify $\overline{\mathbf{MIN\_NSCC}}$ to $\overline{\mathbf{MIN\_NSCC}}$.

See also circuit, link_set_to_booleans.

Key words graph constraint, linear programming, strongly connected component, constraint involving set variables.

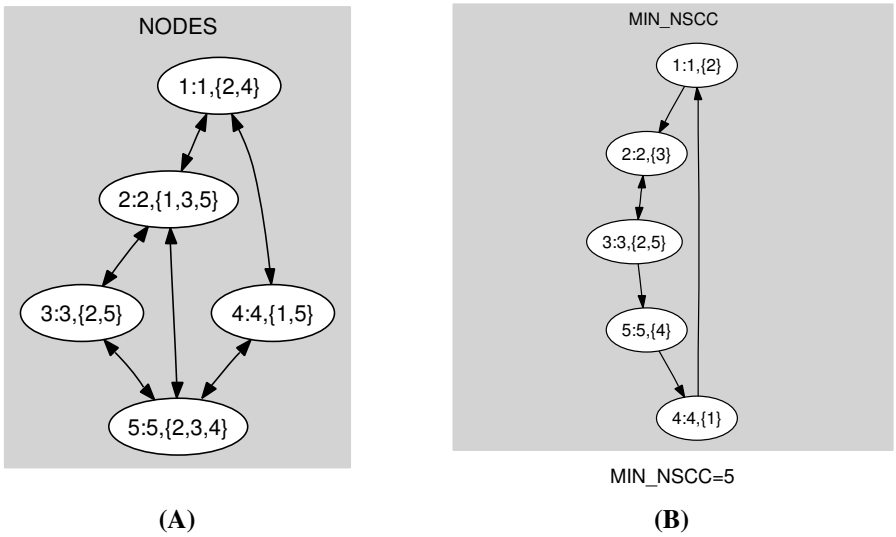


Figure 4.404: Initial and final graph of the `strongly_connected` set constraint

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| ≥ 1
- required(SETS, [ind, set])
- distinct(SETS, ind)
- |CONSTANTS| ≥ 1
- required(CONSTANTS, cst)

Purpose S is equal to the sum of the constants corresponding to the INDEXth set of the SETS collection.

Arc input(s) SETS CONSTANTS

Arc generator $PRODUCT \mapsto$ collection(sets, constants)

Arc arity 2

Arc constraint(s)
- INDEX = sets.ind
- in_set(constants.key, sets.set)

Graph property(ies) **SUM**(CONSTANTS, cst) = S

Example

$$\texttt{sum}\left(\left\langle 8, \left\{\begin{array}{ll}\texttt{ind}-8 & \texttt{set}-\{2,3\},\\ \texttt{ind}-1 & \texttt{set}-\{3\},\\ \texttt{ind}-3 & \texttt{set}-\{1,4,5\},\\ \texttt{ind}-6 & \texttt{set}-\{2,4\}\end{array}\right\},\right.\right.$$
$$\left.\left.\left\{\begin{array}{l}\texttt{cst}-4,\\ \texttt{cst}-9,\\ \texttt{cst}-1,\\ \texttt{cst}-3,\\ \texttt{cst}-1\end{array}\right\}, 10\right\rangle\right)$$

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.

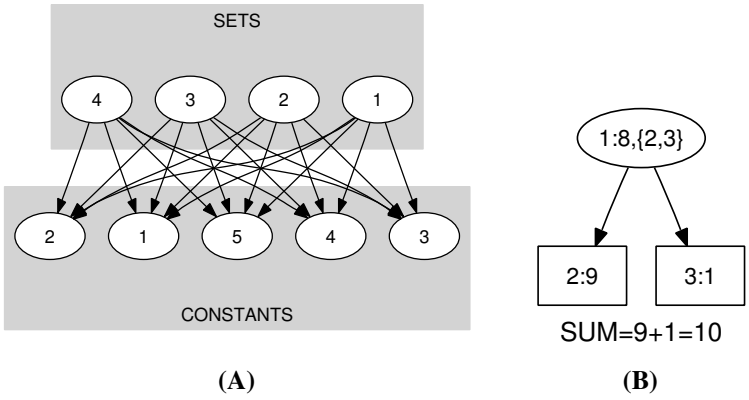


Figure 4.405: Initial and final graph of the `sum` constraint

4.212 sum_ctr

Origin Arithmetic constraint.

Constraint sum_ctr(VARIABLES, CTR, VAR)

Synonym(s) constant_sum.

Argument(s)

VARIABLES	:	collection(var − dvar)
CTR	:	atom
VAR	:	dvar

Restriction(s) required(VARIABLES, var)
CTR ∈ [=, ≠, <, ≥, >, ≤]

Purpose Constraint the sum of a set of domain variables. More precisely let S denotes the sum of the variables of the VARIABLES collection. Enforce the following constraint to hold: S CTR VAR.

Arc input(s) VARIABLES

Arc generator $SELF \mapsto$ collection(variables)

Arc arity 1

Arc constraint(s) TRUE

Graph property(ies) **SUM**(VARIABLES, var) CTR VAR

Example sum_ctr({var − 1, var − 1, var − 4}, =, 6)

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.

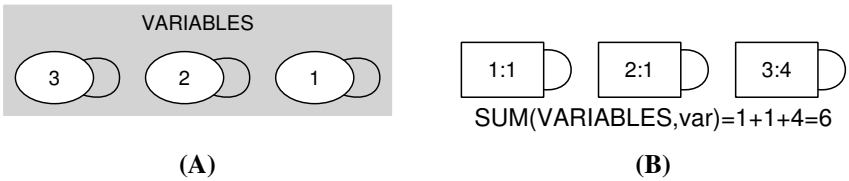


Figure 4.406: Initial and final graph of the sum_ctr constraint

Graph model Since we want to keep all the vertices of the initial graph we use the *SELF* arc generator together with the TRUE arc constraint. This predefined arc constraint allways holds.

Remark When CTR corresponds to = this constraint is referenced under the name constant_sum in KOALOG.

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 VARIABLES collection take a value in the VALUES collection. In addition COST is the sum of the weight attributes associated to the distinct values taken by the variables of VARIABLES.

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)
- $\mathbf{NSOURCE} = |\texttt{VARIABLES}|$
- $\mathbf{SUM}(\texttt{VALUES}, \texttt{weight}) = \texttt{COST}$

Example

$$\texttt{sum\_of\_weights\_of\_distinct\_values}\left(\left\{\begin{array}{l}\texttt{var}-1,\\ \texttt{var}-6,\\ \texttt{var}-1\end{array}\right\}, \left\{\begin{array}{ll}\texttt{val}-1 & \texttt{weight}-5,\\ \texttt{val}-2 & \texttt{weight}-3,\\ \texttt{val}-6 & \texttt{weight}-7\end{array}\right\}, 12\right)$$

Parts (A) and (B) of Figure 4.407 respectively show the initial and final graph. Since we use the $\mathbf{NSOURCE}$ graph property, the source vertices of the final graph are shown in a double circle. Since we also use the $\mathbf{SUM}$ graph property we show the vertices from which we compute the total cost in a box.

Signature Since we use the $PRODUCT$ 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 $\mathbf{NSOURCE} = |\texttt{VARIABLES}|$ to $\mathbf{NSOURCE} \geq |\texttt{VARIABLES}|$ and simplify $\overline{\mathbf{NSOURCE}}$ to $\overline{\mathbf{NSOURCE}}$.

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.

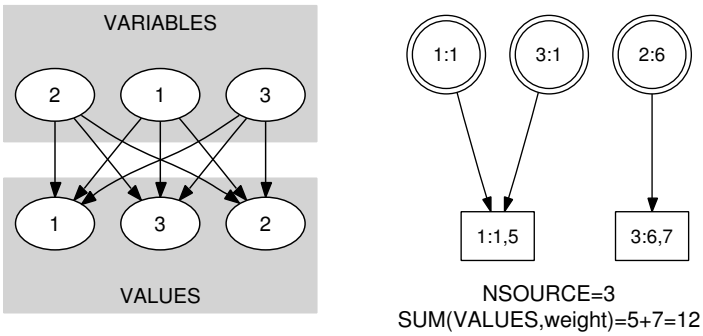


Figure 4.407: Initial and final graph of the `sum_of_weights_of_distinct_values` constraint

4.214 sum_set

Origin H. Cambazard

Constraint sum_set(SV, VALUES, CTR, VAR)

Argument(s)

SV	:	svar
VALUES	:	collection(val − int, coef − int)
CTR	:	atom
VAR	:	dvar

Restriction(s) required(VALUES, [val, coef])
distinct(VALUES, val)
VALUES.coef ≥ 0
CTR $\in [=, \neq, <, \geq, >, \leq]$

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 $SELF \mapsto$ collection(values)

Arc arity 1

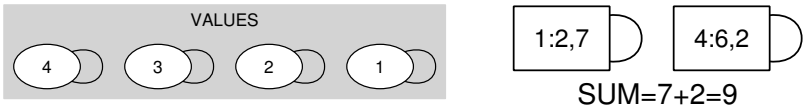
Arc constraint(s) in_set(values.val, SV)

Graph property(ies) **SUM**(VALUES, coef) CTR VAR

Example

$$\texttt{sum\_set}\left(\begin{array}{l}\{2,3,6\},\\ \left\{\begin{array}{ll}\texttt{val}-2 & \texttt{coef}-7,\\ \texttt{val}-9 & \texttt{coef}-1,\\ \texttt{val}-5 & \texttt{coef}-7,\\ \texttt{val}-6 & \texttt{coef}-2\end{array}\right\},=,9\end{array}\right)$$

Parts (A) and (B) of Figure 4.408 respectively show the initial and final graph.



(A) **(B)**

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.

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 $\leq |$NODES$|$
distinct(NODES, index)
NODES.succ ≥ 1
NODES.succ $\leq |$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 $CLIQUE(\neq) \mapsto$ collection(nodes1, nodes2)

Arc arity 2

Arc constraint(s)
- nodes1.succ = nodes2.index
- nodes2.succ = nodes1.index

Graph property(ies) $\mathbf{NARC} = |$NODES$|$

Example symmetric_alldifferent $\left(\left\{ \begin{array}{ll} \text{index} - 1 & \text{succ} - 3, \\ \text{index} - 2 & \text{succ} - 4, \\ \text{index} - 3 & \text{succ} - 1, \\ \text{index} - 4 & \text{succ} - 2 \end{array} \right\} \right)$

Parts (A) and (B) of Figure 4.409 respectively show the initial and final graph. Since we use the $\mathbf{NARC}$ 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 $\mathbf{NARC} = |$NODES$|$ to $\mathbf{NARC} \geq |$NODES$|$ and simplify $\overline{\mathbf{NARC}}$ to $\overline{\mathbf{NARC}}$.

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.

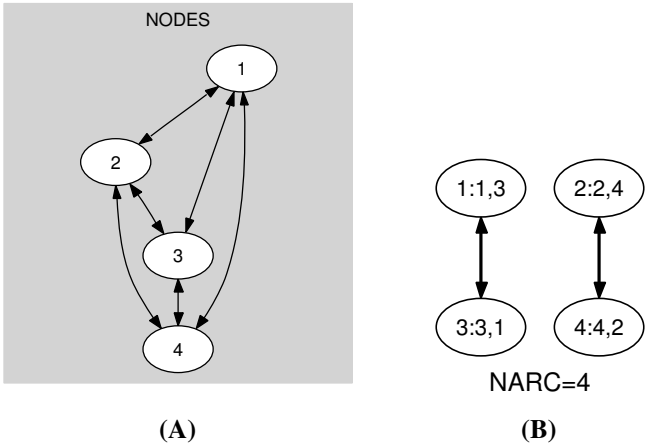


Figure 4.409: Initial and final graph of the `symmetric_alldifferent` constraint

4.216 symmetric_cardinality

Origin Derived from global_cardinality by W. Kocjan.

Constraint symmetric_cardinality(VARS, VALS)

Argument(s)

VARS	:	collection(idvar − int, var − svar, l − int, u − int)
VALS	:	collection(idval − int, val − svar, l − int, u − int)

Restriction(s)

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|

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.

Arc input(s) VARS VALS

Arc generator $PRODUCT \mapsto$ collection(vars, vals)

Arc arity 2

Arc constraint(s)

- 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)

Graph property(ies) NARC = |VARS| ∗ |VALS|

Example

$$\texttt{symmetric\_cardinality}\left(\left\{\begin{array}{llll} \texttt{idvar}-1 & \texttt{var}-\{3\} & \texttt{l}-0 & \texttt{u}-1,\\ \texttt{idvar}-2 & \texttt{var}-\{1\} & \texttt{l}-1 & \texttt{u}-2,\\ \texttt{idvar}-3 & \texttt{var}-\{1,2\} & \texttt{l}-1 & \texttt{u}-2,\\ \texttt{idvar}-4 & \texttt{var}-\{1,3\} & \texttt{l}-2 & \texttt{u}-3 \end{array}\right\},\left\{\begin{array}{llll} \texttt{idval}-1 & \texttt{val}-\{2,3,4\} & \texttt{l}-3 & \texttt{u}-4,\\ \texttt{idval}-2 & \texttt{val}-\{3\} & \texttt{l}-1 & \texttt{u}-1,\\ \texttt{idval}-3 & \texttt{val}-\{1,4\} & \texttt{l}-1 & \texttt{u}-2,\\ \texttt{idval}-4 & \texttt{val}-\emptyset & \texttt{l}-0 & \texttt{u}-1 \end{array}\right\}\right)$$

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.

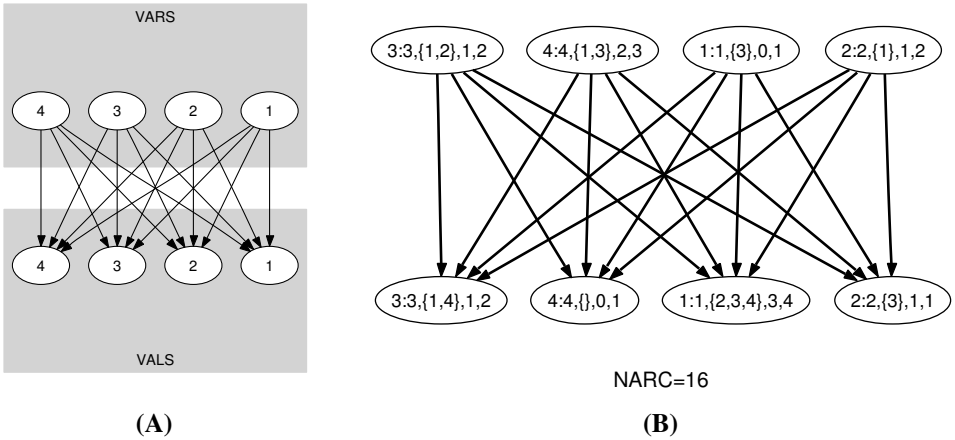


Figure 4.410: Initial and final graph of the `symmetric_cardinality` constraint

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 $|\texttt{VARS}| \cdot |\texttt{VALS}|$. Therefore the maximum number of arcs of the final graph is also equal to $|\texttt{VARS}| \cdot |\texttt{VALS}|$ and we can rewrite $\mathbf{NARC} = |\texttt{VARS}| \cdot |\texttt{VALS}|$ to $\mathbf{NARC} \geq |\texttt{VARS}| \cdot |\texttt{VALS}|$. So we can simplify $\underline{\mathbf{NARC}}$ to $\overline{\mathbf{NARC}}$.

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.

4.217 symmetric_gcc

Origin Derived from global_cardinality by W. Kocjan.

Constraint symmetric_gcc(VARS, VALS)

Synonym(s) sgcc.

Argument(s)

VARS	:	collection(idvar − int, var − svar, nocc − dvar)
VALS	:	collection(idval − int, val − svar, nocc − dvar)

Restriction(s)

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|

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) VARS VALS

Arc generator $PRODUCT \mapsto$ collection(vars, vals)

Arc arity 2

Arc constraint(s)

- in_set(vars.idvar, vals.val) ⇔ in_set(vals.idval, vars.var)
- vars.nocc = card_set(vars.var)
- vals.nocc = card_set(vals.val)

Graph property(ies) NARC = |VARS| ∗ |VALS|

Example

$$\texttt{symmetric\_gcc}\left(\left\{\begin{array}{lll}\texttt{idvar}-1 & \texttt{var}-\{3\} & \texttt{nocc}-1,\\ \texttt{idvar}-2 & \texttt{var}-\{1\} & \texttt{nocc}-1,\\ \texttt{idvar}-3 & \texttt{var}-\{1,2\} & \texttt{nocc}-2,\\ \texttt{idvar}-4 & \texttt{var}-\{1,3\} & \texttt{nocc}-2\end{array}\right\},\left\{\begin{array}{lll}\texttt{idval}-1 & \texttt{val}-\{2,3,4\} & \texttt{nocc}-3,\\ \texttt{idval}-2 & \texttt{val}-\{3\} & \texttt{nocc}-1,\\ \texttt{idval}-3 & \texttt{val}-\{1,4\} & \texttt{nocc}-2,\\ \texttt{idval}-4 & \texttt{val}-\emptyset & \texttt{nocc}-0\end{array}\right\}\right)$$

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.

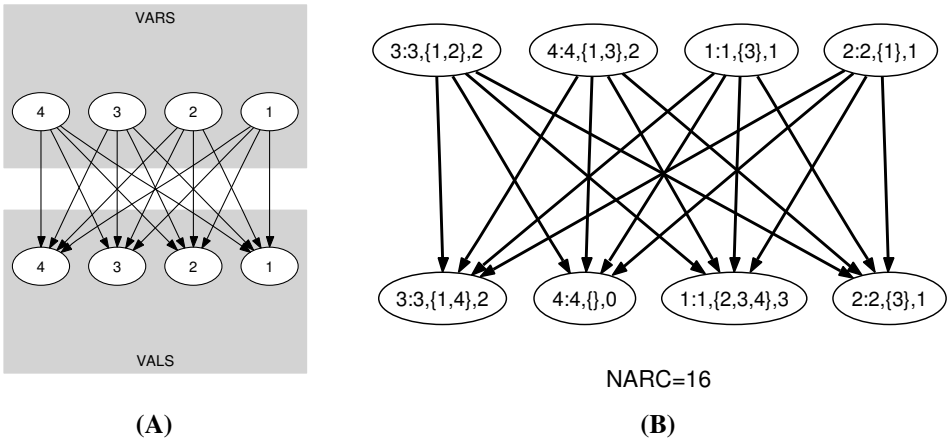


Figure 4.411: Initial and final graph of the `symmetric_gcc` constraint

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 $|\texttt{VARS}| \cdot |\texttt{VALS}|$. Therefore the maximum number of arcs of the final graph is also equal to $|\texttt{VARS}| \cdot |\texttt{VALS}|$ and we can rewrite $\mathbf{NARC} = |\texttt{VARS}| \cdot |\texttt{VALS}|$ to $\mathbf{NARC} \geq |\texttt{VARS}| \cdot |\texttt{VALS}|$. So we can simplify $\underline{\mathbf{NARC}}$ to $\overline{\mathbf{NARC}}$.

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 $\leq$ |NODES|
- required(NODES, [index, succ, start, end])
- |NODES| > 0
- NODES.index ≥ 1
- NODES.index $\leq$ |NODES|
- distinct(NODES, index)
- NODES.succ ≥ 1
- NODES.succ $\leq$ |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 $CLIQUE \mapsto$ collection(nodes1, nodes2)

Arc arity 2

Arc constraint(s)
- 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)
- $\mathbf{MAX\_ID} = 1$
- $\mathbf{NCC} =$ NPATH
- $\mathbf{NVERTEX} =$ |NODES|

Example

$$\texttt{temporal\_path}\left(\begin{array}{l} 2, \left\{\begin{array}{llll} \texttt{index}-1 & \texttt{succ}-2 & \texttt{start}-0 & \texttt{end}-1, \\ \texttt{index}-2 & \texttt{succ}-6 & \texttt{start}-3 & \texttt{end}-5, \\ \texttt{index}-3 & \texttt{succ}-4 & \texttt{start}-0 & \texttt{end}-3, \\ \texttt{index}-4 & \texttt{succ}-5 & \texttt{start}-4 & \texttt{end}-6, \\ \texttt{index}-5 & \texttt{succ}-7 & \texttt{start}-7 & \texttt{end}-8, \\ \texttt{index}-6 & \texttt{succ}-6 & \texttt{start}-7 & \texttt{end}-9, \\ \texttt{index}-7 & \texttt{succ}-7 & \texttt{start}-9 & \texttt{end}-10 \end{array}\right\} \end{array}\right)$$

Parts (A) and (B) of Figure 4.412 respectively show the initial and final graph. Since we use the $\mathbf{MAX\_ID}$, the $\mathbf{NCC}$ and the $\mathbf{NVERTEX}$ 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.

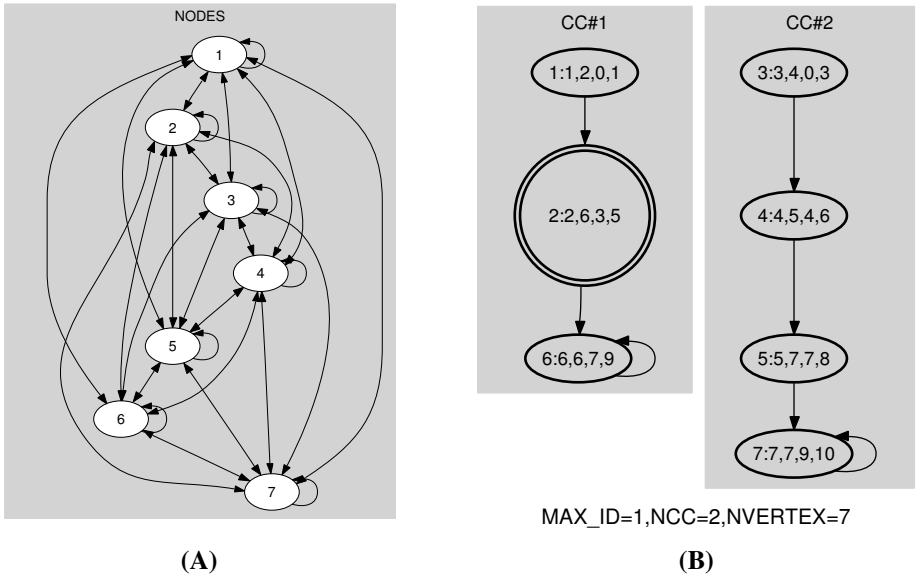


Figure 4.412: Initial and final graph of the `temporal_path` constraint

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 $\mathbf{MAX\_ID} = 1$ 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 $\mathbf{NCC} =$ `NPATH` ensures that we have the required number of paths,
- The third property $\mathbf{NVERTEX} = |$`NODES`$|$ enforces that for each vertex, the start is not located after the end.

Signature

Since we use the graph property $\mathbf{NVERTEX} = |\texttt{NODES}|$ together with the restriction $|\texttt{NODES}| > 0$ the final graph is not empty. Therefore the smallest possible value of $\mathbf{MAX\_ID}$ is equal to 1. So we can rewrite $\mathbf{MAX\_ID} = 1$ to $\mathbf{MAX\_ID} \leq 1$ and simplify $\underline{\overline{\mathbf{MAX\_ID}}}$ to $\underline{\mathbf{MAX\_ID}}$.

Since the maximum number of vertices of the final graph is equal to $|\texttt{NODES}|$ we can rewrite the graph property $\mathbf{NVERTEX} = |\texttt{NODES}|$ to $\mathbf{NVERTEX} \geq |\texttt{NODES}|$ and simplify $\underline{\overline{\mathbf{NVERTEX}}}$ to $\overline{\mathbf{NVERTEX}}$.

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.

4.219 tour

Origin [74]

Constraint tour(NODES)

Synonym(s) atour, cycle.

Argument(s) NODES : collection(index − int, succ − svar)

Restriction(s) |NODES| ≥ 3
required(NODES, [index, succ])
NODES.index ≥ 1
NODES.index ≤ |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 $CLIQUE(\neq) \mapsto$ collection(nodes1, nodes2)

Arc arity 2

Arc constraint(s) in_set(nodes2.index, nodes1.succ) ⇔ in_set(nodes1.index, nodes2.succ)

Graph property(ies) $\mathbf{NARC} = |\text{NODES}| * |\text{NODES}| - |\text{NODES}|$

Arc input(s) NODES

Arc generator $CLIQUE(\neq) \mapsto$ collection(nodes1, nodes2)

Arc arity 2

Arc constraint(s) in_set(nodes2.index, nodes1.succ)

Graph property(ies)
- $\mathbf{MIN\_NSCC} = |\text{NODES}|$
- $\mathbf{MIN\_ID} = 2$
- $\mathbf{MAX\_ID} = 2$
- $\mathbf{MIN\_OD} = 2$
- $\mathbf{MAX\_OD} = 2$

Example

$$\texttt{tour}\left(\left\{\begin{array}{ll}\texttt{index}-1 & \texttt{succ}-\{2,4\},\\ \texttt{index}-2 & \texttt{succ}-\{1,3\},\\ \texttt{index}-3 & \texttt{succ}-\{2,4\},\\ \texttt{index}-4 & \texttt{succ}-\{1,3\}\end{array}\right\}\right)$$

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 |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 |NODES|, we can rewrite the graph property $\underline{\mathbf{MIN\_NSCC}} = |\texttt{NODES}|$ to $\mathbf{MIN\_NSCC} \geq |\texttt{NODES}|$ and simplify $\underline{\mathbf{MIN\_NSCC}}$ to $\mathbf{MIN\_NSCC}$.

See also circuit, cycle, link_set_to_booleans.

Key words graph constraint, undirected graph, Hamiltonian, linear programming, constraint involving set variables.

898 $\overline{\mathbf{NARC}}$, $CLIQUE(\neq)$; $\overline{\underline{\mathbf{MAX\_ID}}}$, $\overline{\underline{\mathbf{MAX\_OD}}}$, $\overline{\underline{\mathbf{MIN\_ID}}}$, $\overline{\underline{\mathbf{MIN\_NSCC}}}$, $\overline{\underline{\mathbf{MIN\_OD}}}$, $CLIQUE(\neq)$

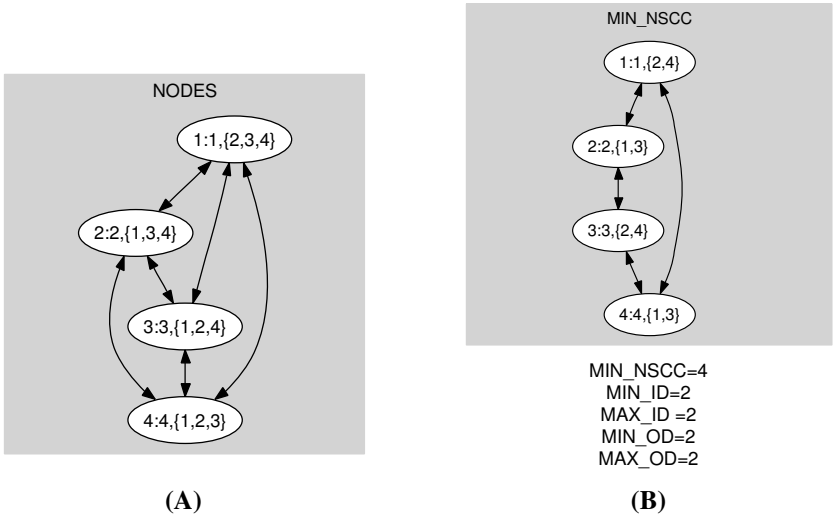


Figure 4.413: Initial and final graph of the `tour` set constraint

4.220 track

Origin [155]

Constraint track(NTRAIL, TASKS)

Argument(s)

NTRAIL	:	int
TASKS	:	collection(trail − int, origin − dvar, end − dvar)

Restriction(s)

NTRAIL $>$ 0
required(TASKS, [trail, origin, end])
TASKS.trail $>$ 0
TASKS.trail $\leq$ NTRAIL

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)

$$\texttt{col}\left(\begin{array}{l}\texttt{TIME\_POINTS} - \texttt{collection}(\texttt{origin} - \texttt{dvar}, \texttt{end} - \texttt{dvar}, \texttt{point} - \texttt{dvar}),\\ \left[\begin{array}{l}\texttt{item}(\texttt{origin} - \texttt{TASKS.origin}, \texttt{end} - \texttt{TASKS.end}, \texttt{point} - \texttt{TASKS.origin}),\\ \texttt{item}(\texttt{origin} - \texttt{TASKS.origin}, \texttt{end} - \texttt{TASKS.end}, \texttt{point} - \texttt{TASKS.end} - 1)\end{array}\right]\end{array}\right)$$

Arc input(s) TASKS

Arc generator $SELF \mapsto$ collection(tasks)

Arc arity 1

Arc constraint(s) tasks.origin $\leq$ tasks.end

Graph property(ies) $\mathbf{NARC} = |$TASKS$|$

Arc input(s) TIME_POINTS TASKS

Arc generator $PRODUCT \mapsto$ collection(time_points, tasks)

Arc arity 2

Arc constraint(s)

- time_points.end $>$ time_points.origin
- tasks.origin $\leq$ time_points.point
- time_points.point $<$ tasks.end

Sets

$$\text{SUCC} \mapsto \left[\begin{array}{l}\texttt{source},\\ \texttt{variables} - \texttt{col}(\texttt{VARIABLES} - \texttt{collection}(\texttt{var} - \texttt{dvar}), [\texttt{item}(\texttt{var} - \texttt{TASKS.trail})])\end{array}\right]$$

Constraint(s) on sets nvalue(NTRAIL, variables)

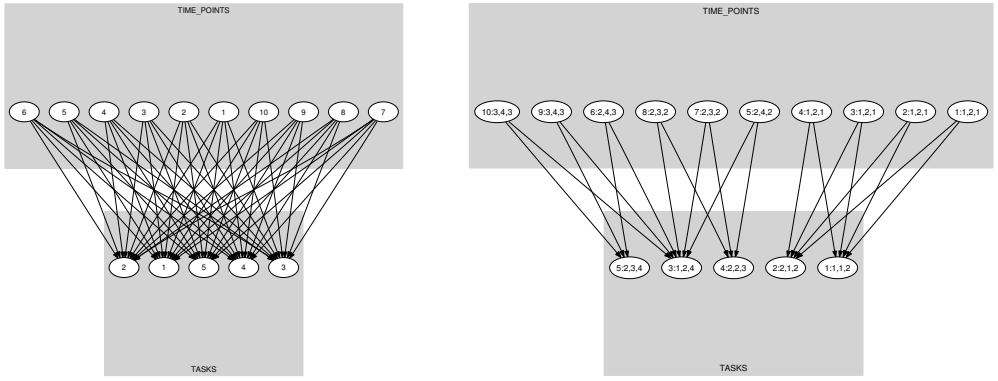
Example

$$\texttt{track}\left(2, \left\{\begin{array}{lll} \texttt{trail}-1 & \texttt{origin}-1 & \texttt{end}-2, \\ \texttt{trail}-2 & \texttt{origin}-1 & \texttt{end}-2, \\ \texttt{trail}-1 & \texttt{origin}-2 & \texttt{end}-4, \\ \texttt{trail}-2 & \texttt{origin}-2 & \texttt{end}-3, \\ \texttt{trail}-2 & \texttt{origin}-3 & \texttt{end}-4 \end{array}\right\}\right)$$

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.



(A) **(B)**

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 $|\texttt{TASKS}|$. Therefore we can rewrite $\mathbf{NARC} = |\texttt{TASKS}|$ to $\mathbf{NARC} \geq |\texttt{TASKS}|$ and simplify $\underline{\overline{\mathbf{NARC}}}$ to $\overline{\mathbf{NARC}}$.

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 ≥ 0
required(NODES, [index, succ])
NODES.index ≥ 1
NODES.index $\leq$ |NODES|
distinct(NODES, index)
NODES.succ ≥ 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 $CLIQUE \mapsto$ collection(nodes1, nodes2)

Arc arity 2

Arc constraint(s) nodes1.succ = nodes2.index

Graph property(ies)
- $\mathbf{MAX\_NSCC} \leq 1$
- $\mathbf{NCC} =$ NTREES

Example

$$\texttt{tree}\left(\begin{array}{l} 2, \left\{\begin{array}{ll} \texttt{index}-1 & \texttt{succ}-1, \\ \texttt{index}-2 & \texttt{succ}-5, \\ \texttt{index}-3 & \texttt{succ}-5, \\ \texttt{index}-4 & \texttt{succ}-7, \\ \texttt{index}-5 & \texttt{succ}-1, \\ \texttt{index}-6 & \texttt{succ}-1, \\ \texttt{index}-7 & \texttt{succ}-7, \\ \texttt{index}-8 & \texttt{succ}-5 \end{array}\right\} \end{array}\right)$$

Parts (A) and (B) of Figure 4.415 respectively show the initial and final graph. Since we use the $\mathbf{NCC}$ 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 NTREES $= \mathbf{NCC} = 2$.

Graph model We use the graph property $\mathbf{MAX\_NSCC} \leq 1$ 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 $\mathbf{NCC} =$ 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.

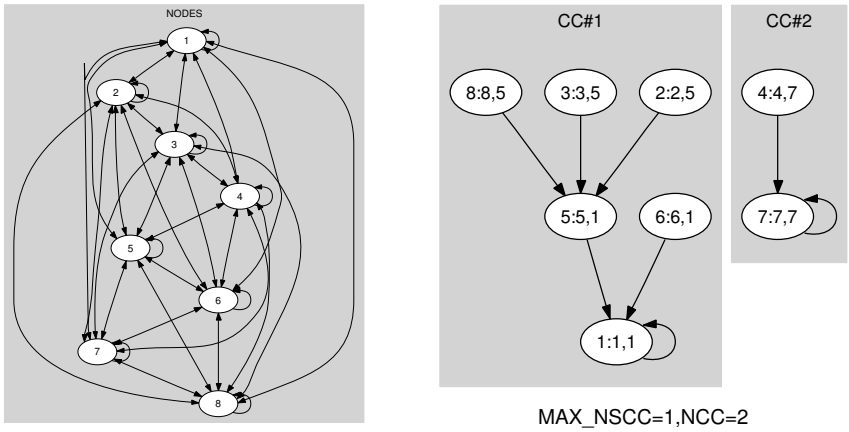


Figure 4.415: Initial and final graph of the `tree` constraint

4.222 tree_range

Origin Derived from tree.

Constraint tree_range(NTREES, R, NODES)

Argument(s)

NTREES	:	dvar
R	:	dvar
NODES	:	collection(index − int, succ − dvar)

Restriction(s)
- NTREES ≥ 0
- R ≥ 0
- R $< |$NODES$|$
- required(NODES, [index, succ])
- NODES.index ≥ 1
- NODES.index $\leq |$NODES$|$
- distinct(NODES, index)
- NODES.succ ≥ 1
- NODES.succ $\leq |$NODES$|$

Purpose 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.

Arc input(s) NODES

Arc generator $CLIQUE \mapsto$ collection(nodes1, nodes2)

Arc arity 2

Arc constraint(s) nodes1.succ = nodes2.index

Graph property(ies)
- **MAX_NSCC** ≤ 1
- **NCC** = NTREES
- **RANGE_DRG** = R

Example

$$\texttt{tree\_range}\left(\begin{array}{c} 2, 1, \left\{\begin{array}{ll} \texttt{index}-1 & \texttt{succ}-1, \\ \texttt{index}-2 & \texttt{succ}-5, \\ \texttt{index}-3 & \texttt{succ}-5, \\ \texttt{index}-4 & \texttt{succ}-7, \\ \texttt{index}-5 & \texttt{succ}-1, \\ \texttt{index}-6 & \texttt{succ}-1, \\ \texttt{index}-7 & \texttt{succ}-7, \\ \texttt{index}-8 & \texttt{succ}-5 \end{array}\right\} \end{array}\right)$$

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.

See also `tree`, `balance`.

Key words graph constraint, graph partitioning constraint, connected component, tree, balanced tree.

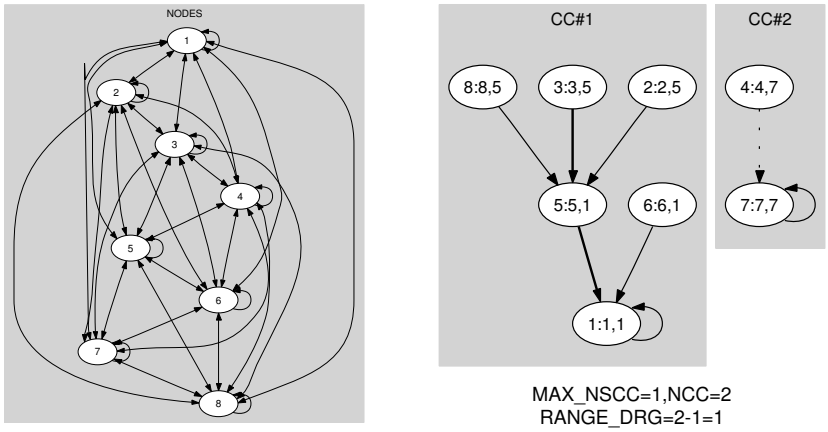


Figure 4.416: Initial and final graph of the `tree_range` constraint

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 $\leq$ |RESOURCE|
distinct(RESOURCE, id)
RESOURCE.nb_task ≥ 0
RESOURCE.nb_task $\leq$ |TASK|
required(TASK, [id, father, resource])
TASK.id $>$ |RESOURCE|
TASK.id $\leq$ |RESOURCE| + |TASK|
distinct(TASK, id)
TASK.father ≥ 1
TASK.father $\leq$ |RESOURCE| + |TASK|
TASK.resource ≥ 1
TASK.resource $\leq$ |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 *resource* vertex and several *task* 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)

$$\texttt{col}\left(\begin{array}{l}\texttt{RESOURCE\_TASK} - \texttt{collection}(\texttt{index} - \texttt{int}, \texttt{succ} - \texttt{dvar}, \texttt{name} - \texttt{dvar}),\\ \left[\begin{array}{l}\texttt{item}(\texttt{index} - \texttt{RESOURCE.id}, \texttt{succ} - \texttt{RESOURCE.id}, \texttt{name} - \texttt{RESOURCE.id}),\\ \texttt{item}(\texttt{index} - \texttt{TASK.id}, \texttt{succ} - \texttt{TASK.father}, \texttt{name} - \texttt{TASK.resource})\end{array}\right]\end{array}\right)$$

Arc input(s) RESOURCE_TASK

Arc generator $CLIQUE \mapsto$ collection(resource_task1, resource_task2)

Arc arity 2

Arc constraint(s)

- resource_task1.succ = resource_task2.index
- resource_task1.name = resource_task2.name

Graph property(ies)

- $\mathbf{MAX\_NSCC} \leq 1$
- $\mathbf{NCC} =$ |RESOURCE|
- $\mathbf{NVERTEX} =$ |RESOURCE| + |TASK|

For all items of RESOURCE:

Arc input(s) RESOURCE_TASK

Arc generator $CLIQUE \mapsto \texttt{collection}(\texttt{resource\_task1}, \texttt{resource\_task2})$

Arc arity 2

Arc constraint(s)
- resource_task1.succ = resource_task2.index
- resource_task1.name = resource_task2.name
- resource_task1.name = RESOURCE.id

Graph property(ies) $\mathbf{NVERTEX} = \texttt{RESOURCE.nb\_task} + 1$

Example
$$\texttt{tree\_resource}\left(\begin{array}{l}\left\{\begin{array}{llll}\texttt{id}-1 & \texttt{nb\_task}-4,\\ \texttt{id}-2 & \texttt{nb\_task}-0,\\ \texttt{id}-3 & \texttt{nb\_task}-1\end{array}\right\},\\ \left\{\begin{array}{llll}\texttt{id}-4 & \texttt{father}-8 & \texttt{resource}-1,\\ \texttt{id}-5 & \texttt{father}-3 & \texttt{resource}-3,\\ \texttt{id}-6 & \texttt{father}-8 & \texttt{resource}-1,\\ \texttt{id}-7 & \texttt{father}-1 & \texttt{resource}-1,\\ \texttt{id}-8 & \texttt{father}-1 & \texttt{resource}-1\end{array}\right\}\end{array}\right)$$

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 |RESOURCE| + |TASK| vertices, the corresponding final graph cannot have more than |RESOURCE| + |TASK| vertices. Therefore we can rewrite the graph property $\mathbf{NVERTEX} = |\texttt{RESOURCE}| + |\texttt{TASK}|$ to $\mathbf{NVERTEX} \geq |\texttt{RESOURCE}| + |\texttt{TASK}|$ and simplify $\overline{\mathbf{NVERTEX}}$ to $\mathbf{NVERTEX}$.

See also tree.

Key words graph constraint, tree, resource constraint, graph partitioning constraint, connected component, derived collection.

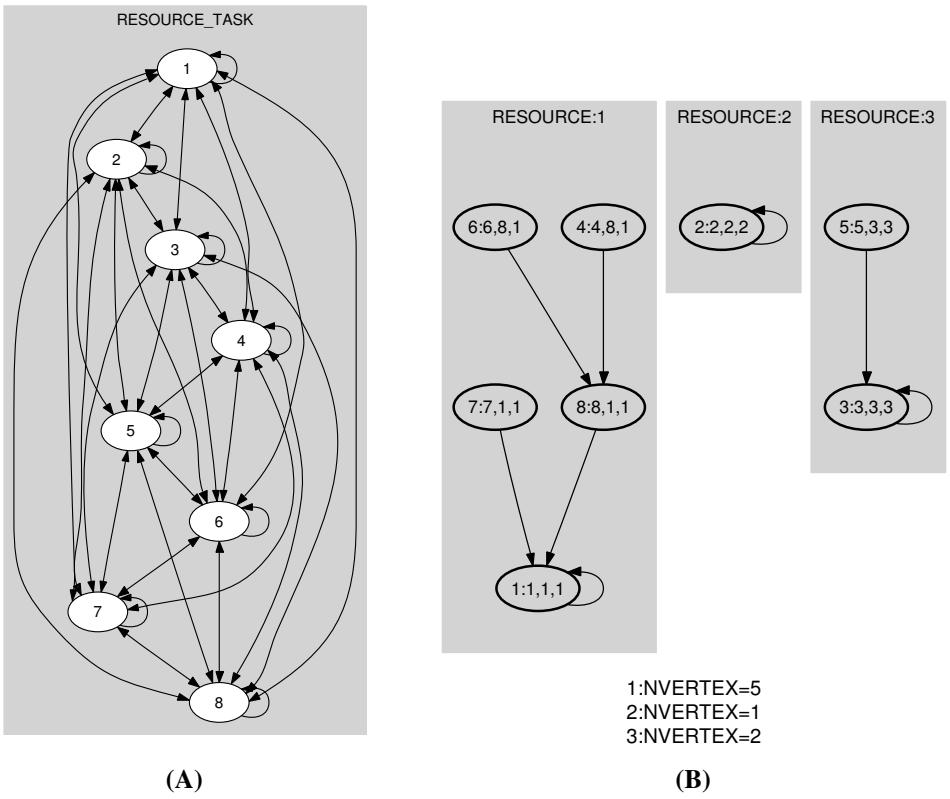


Figure 4.417: Initial and final graph of the `tree_resource` constraint

4.224 two_layer_edge_crossing

Origin Inspired by [157].

Constraint two_layer_edge_crossing(NCROSS, VERTICES_LAYER1, VERTICES_LAYER2, EDGES)

Argument(s)

NCROSS	:	dvar
VERTICES_LAYER1	:	collection(id − int, pos − dvar)
VERTICES_LAYER2	:	collection(id − int, pos − dvar)
EDGES	:	collection(id − int, vertex1 − int, vertex2 − int)

Restriction(s)

NCROSS ≥ 0
required(VERTICES_LAYER1, [id, pos])
VERTICES_LAYER1.id ≥ 1
VERTICES_LAYER1.id $\leq$ |VERTICES_LAYER1|
distinct(VERTICES_LAYER1, id)
required(VERTICES_LAYER2, [id, pos])
VERTICES_LAYER2.id ≥ 1
VERTICES_LAYER2.id $\leq$ |VERTICES_LAYER2|
distinct(VERTICES_LAYER2, id)
required(EDGES, [id, vertex1, vertex2])
EDGES.id ≥ 1
EDGES.id $\leq$ |EDGES|
distinct(EDGES, id)
EDGES.vertex1 ≥ 1
EDGES.vertex1 $\leq$ |VERTICES_LAYER1|
EDGES.vertex2 ≥ 1
EDGES.vertex2 $\leq$ |VERTICES_LAYER2|

Purpose NCROSS is the number of line-segments intersections.

Derived Collection(s)

$$\texttt{col}\left(\begin{array}{l}\texttt{EDGES\_EXTREMITIES} - \texttt{collection}(\texttt{layer1} - \texttt{dvar}, \texttt{layer2} - \texttt{dvar}),\\ \left[\ \texttt{item}\left(\begin{array}{l}\texttt{layer1} - \texttt{EDGES.vertex1}(\texttt{VERTICES\_LAYER1}, \texttt{pos}, \texttt{id}),\\ \texttt{layer2} - \texttt{EDGES.vertex2}(\texttt{VERTICES\_LAYER2}, \texttt{pos}, \texttt{id})\end{array}\right)\ \right]\end{array}\right)$$

Arc input(s) EDGES_EXTREMITIES

Arc generator $CLIQUE(<) \mapsto$ collection(edges_extremities1, edges_extremities2)

Arc arity 2

Arc constraint(s)

$$\bigvee\left(\begin{array}{l}\bigwedge\left(\begin{array}{l}\texttt{edges\_extremities1.layer1} < \texttt{edges\_extremities2.layer1},\\ \texttt{edges\_extremities1.layer2} > \texttt{edges\_extremities2.layer2}\end{array}\right),\\ \bigwedge\left(\begin{array}{l}\texttt{edges\_extremities1.layer1} > \texttt{edges\_extremities2.layer1},\\ \texttt{edges\_extremities1.layer2} < \texttt{edges\_extremities2.layer2}\end{array}\right)\end{array}\right)$$

Graph property(ies) $\mathbf{NARC} = \texttt{NCROSS}$

Example

$$\texttt{two\_layer\_edge\_crossing}\left(\begin{array}{l} 2,\{\texttt{id}-1\ \texttt{pos}-1,\texttt{id}-2\ \texttt{pos}-2\}, \\ \left\{\begin{array}{lll}\texttt{id}-1 & \texttt{pos}-3,\\ \texttt{id}-2 & \texttt{pos}-1,\\ \texttt{id}-3 & \texttt{pos}-2\end{array}\right\}, \\ \left\{\begin{array}{lll}\texttt{id}-1 & \texttt{vertex1}-2 & \texttt{vertex2}-2,\\ \texttt{id}-2 & \texttt{vertex1}-2 & \texttt{vertex2}-3,\\ \texttt{id}-3 & \texttt{vertex1}-1 & \texttt{vertex2}-1\end{array}\right\} \end{array}\right)$$

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.

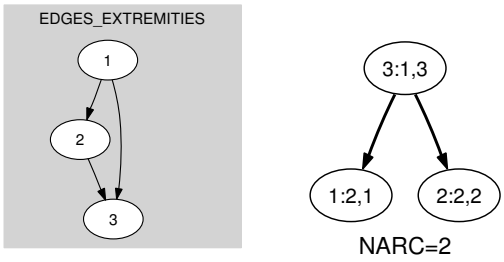


Figure 4.418: Initial and final graph of the `two_layer_edge_crossing` constraint

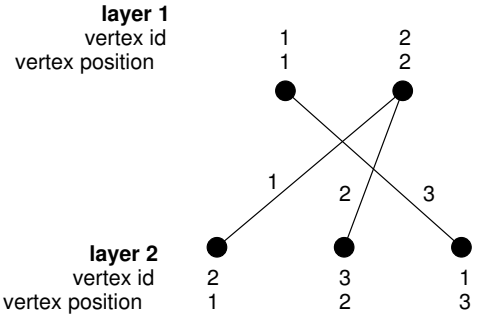


Figure 4.419: Intersection between line-segments joining two layers

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.

4.225 two_orth_are_in_contact

Origin Used for defining orths_are_connected.

Constraint two_orth_are_in_contact(ORTHOTOPE1, ORTHOTOPE2)

Type(s) ORTHOTOPE : collection(ori − dvar, siz − dvar, end − dvar)

Argument(s)

ORTHOTOPE1	:	ORTHOTOPE
ORTHOTOPE2	:	ORTHOTOPE

Restriction(s)
- |ORTHOTOPE| > 0
- require_at_least(2, ORTHOTOPE, [ori, siz, end])
- ORTHOTOPE.siz > 0
- |ORTHOTOPE1| = |ORTHOTOPE2|
- orth_link_ori_siz_end(ORTHOTOPE1)
- orth_link_ori_siz_end(ORTHOTOPE2)

Purpose

Enforce the following conditions on two orthotopes O_1 and O_2:
- 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) ORTHOTOPE1 ORTHOTOPE2

Arc generator $PRODUCT(=) \mapsto$ collection(orthotope1, orthotope2)

Arc arity 2

Arc constraint(s)
- orthotope1.end > orthotope2.ori
- orthotope2.end > orthotope1.ori

Graph property(ies) $\mathbf{NARC} = |\text{ORTHOTOPE1}| - 1$

Arc input(s) ORTHOTOPE1 ORTHOTOPE2

Arc generator $PRODUCT(=) \mapsto$ collection(orthotope1, orthotope2)

Arc arity 2

Arc constraint(s) $\max\left(\ 0, \max(\text{orthotope1.ori}, \text{orthotope2.ori}) - \min(\text{orthotope1.end}, \text{orthotope2.end})\ \right) = 0$

Graph property(ies) $\mathbf{NARC} = |\text{ORTHOTOPE1}|$

Example

$$\texttt{two\_orth\_are\_in\_contact}\left(\left\{\begin{array}{lll}\texttt{ori}-1 & \texttt{siz}-3 & \texttt{end}-4,\\ \texttt{ori}-5 & \texttt{siz}-2 & \texttt{end}-7\end{array}\right\},\left\{\begin{array}{lll}\texttt{ori}-3 & \texttt{siz}-2 & \texttt{end}-5,\\ \texttt{ori}-2 & \texttt{siz}-3 & \texttt{end}-5\end{array}\right\}\right)$$

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 $\mathbf{NARC}$ 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.

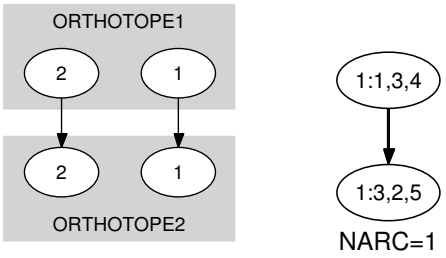


Figure 4.420: Initial and final graph of the `two_orth_are_in_contact` constraint

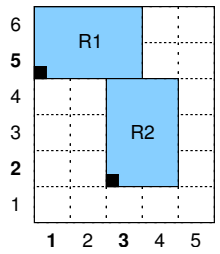


Figure 4.421: Two connected rectangles

Signature

Consider the second graph constraint. Since we use the arc generator $PRODUCT(=)$ on the collections ORTHOTOPE1 and ORTHOTOPE2, and because of the restriction $|\texttt{ORTHOTOPE1}| = |\texttt{ORTHOTOPE2}|$, the maximum number of arcs of the corresponding final graph is equal to $|\texttt{ORTHOTOPE1}|$. Therefore we can rewrite the graph property $\mathbf{NARC} = |\texttt{ORTHOTOPE1}|$ to $\mathbf{NARC} \geq |\texttt{ORTHOTOPE1}|$ and simplify $\underline{\mathbf{NARC}}$ to $\overline{\mathbf{NARC}}$.

Automaton

Figure 4.422 depicts the automaton associated to the `two_orth_are_in_contact` constraint. Let $\texttt{ORI1}_i$, $\texttt{SIZ1}_i$ and $\texttt{END1}_i$ respectively be the ori, the siz and the end attributes of the i^{th} item of the ORTHOTOPE1 collection. Let $\texttt{ORI2}_i$, $\texttt{SIZ2}_i$ and $\texttt{END2}_i$ respectively be the ori, the siz and the end attributes of the i^{th} item of the ORTHOTOPE2 collection. To each sextuple $(\texttt{ORI1}_i, \texttt{SIZ1}_i, \texttt{END1}_i, \texttt{ORI2}_i, \texttt{SIZ2}_i, \texttt{END2}_i)$ corresponds a signature variable $\texttt{S}_i$, which takes its value in $\{0, 1, 2\}$, as well as the following signature constraint:

$$((\texttt{SIZ1}_i > 0) \wedge (\texttt{SIZ2}_i > 0) \wedge (\texttt{END1}_i > \texttt{ORI2}_i) \wedge (\texttt{END2}_i > \texttt{ORI1}_i)) \Leftrightarrow \texttt{S}_i = 0$$

$$((\texttt{SIZ1}_i > 0) \wedge (\texttt{SIZ2}_i > 0) \wedge (\texttt{END1}_i = \texttt{ORI2}_i \vee \texttt{END2}_i = \texttt{ORI1}_i)) \Leftrightarrow \texttt{S}_i = 1.$$

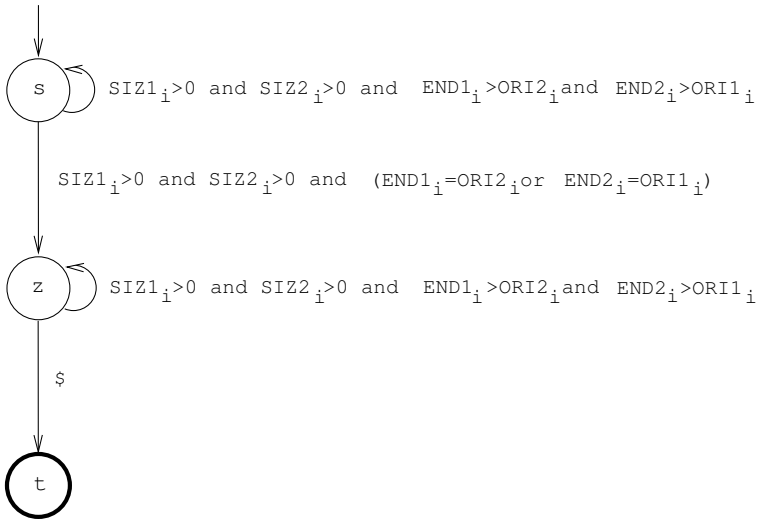


Figure 4.422: Automaton of the `two_orth_are_in_contact` constraint

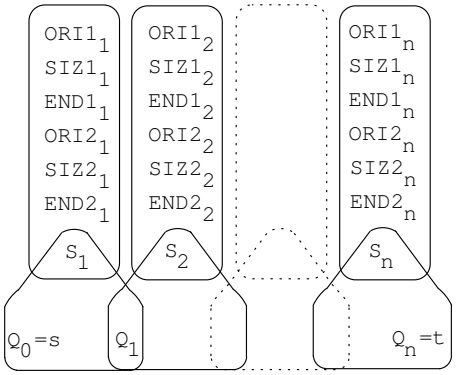


Figure 4.423: Hypergraph of the reformulation corresponding to the automaton of the `two_orth_are_in_contact` constraint

Used in `orths_are_connected`.

Key words geometrical constraint, touch, contact, non-overlapping, orthotope, Berge-acyclic constraint network, automaton, automaton without counters.

4.226 two_orth_column

Origin Used for defining diffn_column.

Constraint two_orth_column(ORTHOTOPE1, ORTHOTOPE2, N)

Type(s) ORTHOTOPE : collection(ori − dvar, siz − dvar, end − dvar)

Argument(s)

ORTHOTOPE1	:	ORTHOTOPE
ORTHOTOPE2	:	ORTHOTOPE
N	:	int

Restriction(s)
$|\texttt{ORTHOTOPE}| > 0$
$\texttt{require\_at\_least}(2, \texttt{ORTHOTOPE}, [\texttt{ori}, \texttt{siz}, \texttt{end}])$
$\texttt{ORTHOTOPE.siz} \geq 0$
$|\texttt{ORTHOTOPE1}| = |\texttt{ORTHOTOPE2}|$
$\texttt{orth\_link\_ori\_siz\_end}(\texttt{ORTHOTOPE1})$
$\texttt{orth\_link\_ori\_siz\_end}(\texttt{ORTHOTOPE2})$
$\texttt{N} > 0$
$\texttt{N} \leq |\texttt{ORTHOTOPE1}|$

Purpose □

Arc input(s) ORTHOTOPE1 ORTHOTOPE2

Arc generator $PRODUCT(=) \mapsto \texttt{collection}(\texttt{orthotope1}, \texttt{orthotope2})$

Arc arity 2

Arc constraint(s)

$$\bigwedge \left(\begin{array}{l} \texttt{orthotope1.key} = \texttt{N}, \\ \texttt{orthotope1.ori} < \texttt{orthotope2.end}, \\ \texttt{orthotope2.ori} < \texttt{orthotope1.end}, \\ \texttt{orthotope1.siz} > 0, \\ \texttt{orthotope2.siz} > 0 \end{array} \right) \Rightarrow$$
$$\bigwedge \left(\begin{array}{l} \texttt{min}(\texttt{orthotope1.end}, \texttt{orthotope2.end}) - \texttt{max}(\texttt{orthotope1.ori}, \texttt{orthotope2.ori}) = \\ \texttt{orthotope1.siz} \\ \texttt{orthotope1.siz} = \texttt{orthotope2.siz} \end{array} \right),$$

Graph property(ies) $\mathbf{NARC} = 1$

Example

$$\texttt{two\_orth\_column}\left(\left\{ \begin{array}{llll} \texttt{ori} - 1 & \texttt{siz} - 3 & \texttt{end} - 4, \\ \texttt{ori} - 1 & \texttt{siz} - 1 & \texttt{end} - 2 \end{array} \right\}, \left\{ \begin{array}{llll} \texttt{ori} - 4 & \texttt{siz} - 2 & \texttt{end} - 6, \\ \texttt{ori} - 1 & \texttt{siz} - 3 & \texttt{end} - 4 \end{array} \right\}, 1 \right)$$

Used in diffn_column.

See also diffn.

Key words geometrical constraint, positioning constraint, orthotope, guillotine cut.

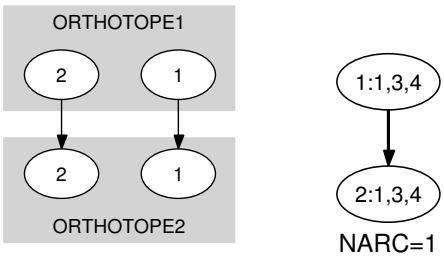


Figure 4.424: Initial and final graph of the `two_orth_column` constraint

4.227 two_orth_do_not_overlap

Origin Used for defining diffn.

Constraint two_orth_do_not_overlap(ORTHOTOPE1, ORTHOTOPE2)

Type(s) ORTHOTOPE : collection(ori − dvar, siz − dvar, end − dvar)

Argument(s)

ORTHOTOPE1	:	ORTHOTOPE
ORTHOTOPE2	:	ORTHOTOPE

Restriction(s)
- |ORTHOTOPE| > 0
- require_at_least(2, ORTHOTOPE, [ori, siz, end])
- ORTHOTOPE.siz ≥ 0
- |ORTHOTOPE1| = |ORTHOTOPE2|
- orth_link_ori_siz_end(ORTHOTOPE1)
- orth_link_ori_siz_end(ORTHOTOPE2)

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) ORTHOTOPE1 ORTHOTOPE2

Arc generator $SYMMETRIC\_PRODUCT(=) \mapsto$ collection(orthotope1, orthotope2)

Arc arity 2

Arc constraint(s) orthotope1.end ≤ orthotope2.ori ∨ orthotope1.siz = 0

Graph property(ies) $\mathbf{NARC} \geq 1$

Example

$$\texttt{two\_orth\_do\_not\_overlap}\left(\left\{\begin{array}{lll}\texttt{ori}-2 & \texttt{siz}-2 & \texttt{end}-4,\\ \texttt{ori}-1 & \texttt{siz}-3 & \texttt{end}-4\end{array}\right\},\left\{\begin{array}{lll}\texttt{ori}-4 & \texttt{siz}-4 & \texttt{end}-8,\\ \texttt{ori}-3 & \texttt{siz}-3 & \texttt{end}-6\end{array}\right\}\right)$$

Parts (A) and (B) of Figure 4.425 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. 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.

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 $\texttt{ORI1}_i$, $\texttt{SIZ1}_i$ and $\texttt{END1}_i$ respectively be the `ori`, the `siz` and the `end` attributes of the i^{th} item of the `ORTHOTOPE1` collection. Let $\texttt{ORI2}_i$, $\texttt{SIZ2}_i$ and $\texttt{END2}_i$ respectively be the `ori`, the `siz` and the `end` attributes of the i^{th} item of the `ORTHOTOPE2` collection. To each sextuple $(\texttt{ORI1}_i, \texttt{SIZ1}_i, \texttt{END1}_i, \texttt{ORI2}_i, \texttt{SIZ2}_i, \texttt{END2}_i)$ corresponds a 0-1 signature variable $\texttt{S}_i$ as well as the following signature constraint: $((\texttt{SIZ1}_i > 0) \wedge (\texttt{SIZ2}_i > 0) \wedge (\texttt{END1}_i > \texttt{ORI2}_i) \wedge (\texttt{END2}_i > \texttt{ORI1}_i)) \Leftrightarrow \texttt{S}_i$.

Used in `diffn`.

Key words geometrical constraint, non-overlapping, orthotope, Berge-acyclic constraint network, automaton, automaton without counters.

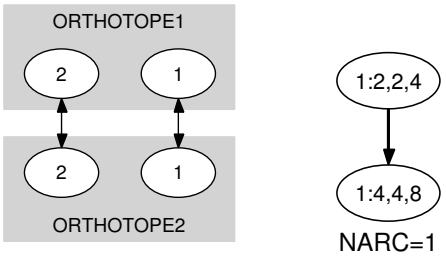


Figure 4.425: Initial and final graph of the `two_orth_do_not_overlap` constraint

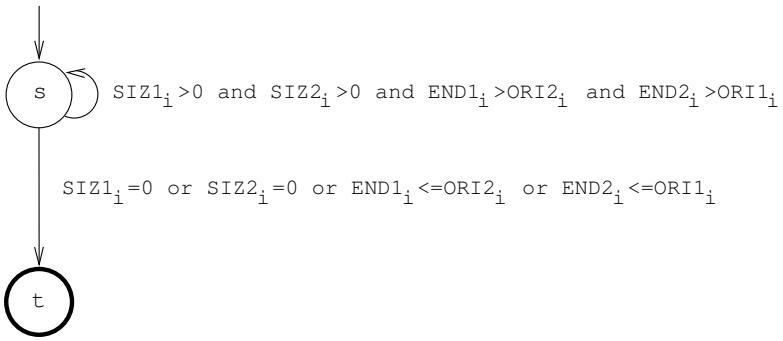


Figure 4.426: Automaton of the two_orth_do_not_overlap constraint

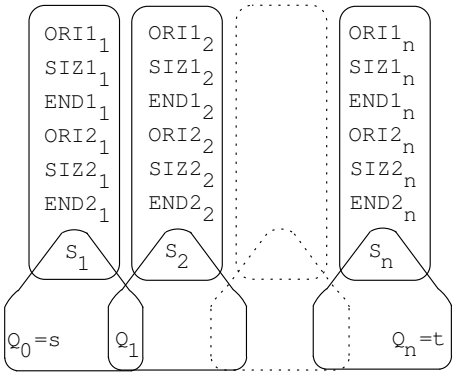


Figure 4.427: Hypergraph of the reformulation corresponding to the automaton of the two_orth_do_not_overlap constraint

4.228 two_orth_include

Origin Used for defining diffn_include.

Constraint two_orth_include(ORTHOTOPE1, ORTHOTOPE2, N)

Type(s) ORTHOTOPE : collection(ori − dvar, siz − dvar, end − dvar)

Argument(s)

ORTHOTOPE1	:	ORTHOTOPE
ORTHOTOPE2	:	ORTHOTOPE
N	:	int

Restriction(s)
- $|\texttt{ORTHOTOPE}| > 0$
- $\texttt{require\_at\_least}(2, \texttt{ORTHOTOPE}, [\texttt{ori}, \texttt{siz}, \texttt{end}])$
- $\texttt{ORTHOTOPE.siz} \geq 0$
- $|\texttt{ORTHOTOPE1}| = |\texttt{ORTHOTOPE2}|$
- $\texttt{orth\_link\_ori\_siz\_end}(\texttt{ORTHOTOPE1})$
- $\texttt{orth\_link\_ori\_siz\_end}(\texttt{ORTHOTOPE2})$
- $\texttt{N} > 0$
- $\texttt{N} \leq |\texttt{ORTHOTOPE1}|$

Purpose □

Arc input(s) ORTHOTOPE1 ORTHOTOPE2

Arc generator $PRODUCT(=) \mapsto \texttt{collection}(\texttt{orthotope1}, \texttt{orthotope2})$

Arc arity 2

Arc constraint(s)

$$\bigwedge \left(\begin{array}{l} \texttt{orthotope1.key} = \texttt{N}, \\ \texttt{orthotope1.ori} < \texttt{orthotope2.end}, \\ \texttt{orthotope2.ori} < \texttt{orthotope1.end}, \\ \texttt{orthotope1.siz} > 0, \\ \texttt{orthotope2.siz} > 0 \end{array} \right) \Rightarrow \bigvee \left(\begin{array}{l} \texttt{min}(\texttt{orthotope1.end}, \texttt{orthotope2.end}) - \texttt{max}(\texttt{orthotope1.ori}, \texttt{orthotope2.ori}) = \\ \texttt{orthotope1.siz} \\ \texttt{min}(\texttt{orthotope1.end}, \texttt{orthotope2.end}) - \texttt{max}(\texttt{orthotope1.ori}, \texttt{orthotope2.ori}) = \\ \texttt{orthotope2.siz} \end{array} \right),$$

Graph property(ies) $\mathbf{NARC} = 1$

Example

$$\texttt{two\_orth\_include} \left(\left\{ \begin{array}{lll} \texttt{ori} - 1 & \texttt{siz} - 3 & \texttt{end} - 4, \\ \texttt{ori} - 1 & \texttt{siz} - 1 & \texttt{end} - 2 \end{array} \right\}, \left\{ \begin{array}{lll} \texttt{ori} - 1 & \texttt{siz} - 2 & \texttt{end} - 3, \\ \texttt{ori} - 2 & \texttt{siz} - 3 & \texttt{end} - 5 \end{array} \right\}, 1 \right)$$

Used in diffn_include.

See also `diffn`.

Key words geometrical constraint, positioning constraint, orthotope.

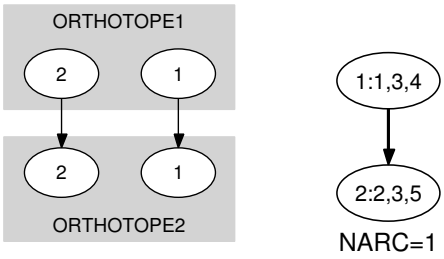


Figure 4.428: Initial and final graph of the `two_orth_include` constraint

4.229 used_by

Origin N. Beldiceanu

Constraint used_by(VARIABLES1, VARIABLES2)

Argument(s)

VARIABLES1	:	collection(var − dvar)
VARIABLES2	:	collection(var − dvar)

Restriction(s)
|VARIABLES1| ≥ |VARIABLES2|
required(VARIABLES1, var)
required(VARIABLES2, var)

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 $PRODUCT \mapsto$ collection(variables1, variables2)

Arc arity 2

Arc constraint(s) variables1.var = variables2.var

Graph property(ies)
- for all connected components: $\mathbf{NSOURCE} \geq \mathbf{NSINK}$
- $\mathbf{NSINK} = |\mathtt{VARIABLES2}|$

Example

$$\mathtt{used\_by}\left(\left\{\begin{array}{l}\mathtt{var}-1,\\ \mathtt{var}-9,\\ \mathtt{var}-1,\\ \mathtt{var}-5,\\ \mathtt{var}-2,\\ \mathtt{var}-1\end{array}\right\},\ \{\mathtt{var}-1,\mathtt{var}-1,\mathtt{var}-2,\mathtt{var}-5\}\right)$$

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 |VARIABLES2|. Therefore we can rewrite $\mathbf{NSINK} = |\texttt{VARIABLES2}|$ to $\mathbf{NSINK} \geq |\texttt{VARIABLES2}|$ and simplify $\underline{\overline{\mathbf{NSINK}}}$ to $\overline{\mathbf{NSINK}}$.

Automaton Figure 4.430 depicts the automaton associated to the used_by constraint. To each item of the collection VARIABLES1 corresponds a signature variable $\texttt{S}_i$, which is equal to 0. To each item of the collection VARIABLES2 corresponds a signature variable $\texttt{S}_{i+|\texttt{VARIABLES1}|}$, 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.

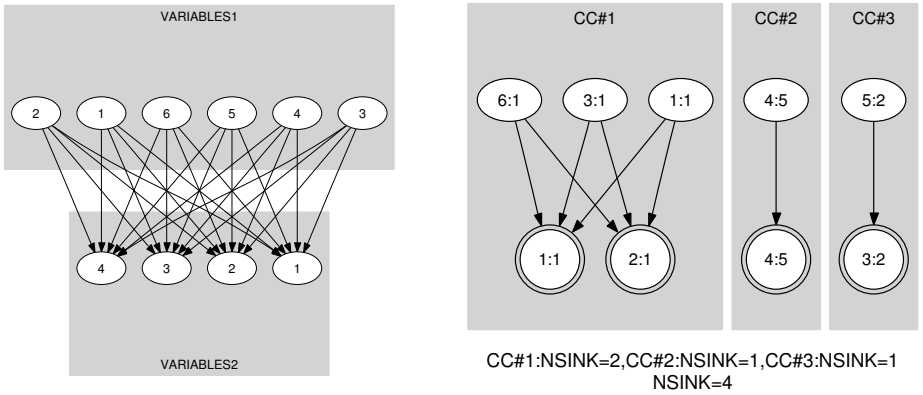


Figure 4.429: Initial and final graph of the used_by constraint

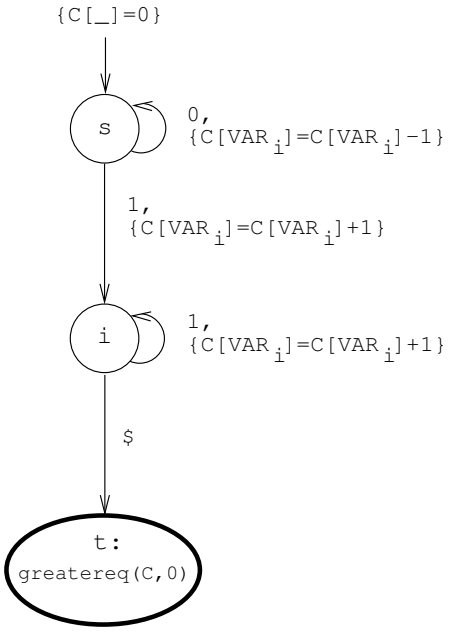


Figure 4.430: Automaton of the used_by constraint

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) |VARIABLES1| ≥ |VARIABLES2|
required(VARIABLES1, var)
required(VARIABLES2, var)
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 [SIZE_INTERVAL $\cdot\, i$, SIZE_INTERVAL $\cdot\, i +$ SIZE_INTERVAL $- 1$]. For all integer i we have $M_i > 0 \Rightarrow N_i > 0$.

Arc input(s) VARIABLES1 VARIABLES2

Arc generator $PRODUCT \mapsto$ collection(variables1, variables2)

Arc arity 2

Arc constraint(s) variables1.var/SIZE_INTERVAL = variables2.var/SIZE_INTERVAL

Graph property(ies)
- for all connected components: $\mathbf{NSOURCE} \geq \mathbf{NSINK}$
- $\mathbf{NSINK} = |\text{VARIABLES2}|$

Example

$$\text{used\_by\_interval}\left(\left\{\begin{array}{l}\text{var}-1,\\ \text{var}-9,\\ \text{var}-1,\\ \text{var}-8,\\ \text{var}-6,\\ \text{var}-2\end{array}\right\},\left\{\begin{array}{l}\text{var}-1,\\ \text{var}-0,\\ \text{var}-7,\\ \text{var}-7\end{array}\right\},3\right)$$

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 $\mathbf{NSOURCE}$ and $\mathbf{NSINK}$ 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 |VARIABLES2|.

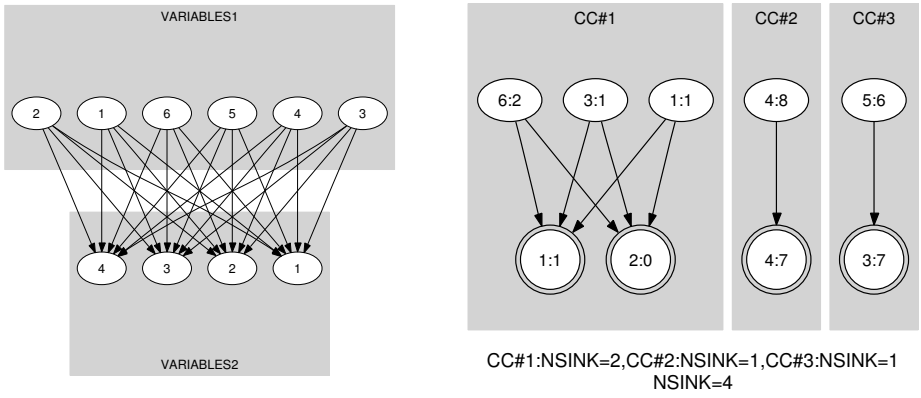


Figure 4.431: Initial and final graph of the used_by_interval constraint

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 |VARIABLES2|. Therefore we can rewrite $\mathbf{NSINK} = |\texttt{VARIABLES2}|$ to $\mathbf{NSINK} \geq |\texttt{VARIABLES2}|$ and simplify $\underline{\overline{\mathbf{NSINK}}}$ to $\overline{\mathbf{NSINK}}$.

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) |VARIABLES1| ≥ |VARIABLES2|
required(VARIABLES1, var)
required(VARIABLES2, var)
M > 0

Purpose For each integer R in $[0, \mathtt{M}-1]$, 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 $[0, \mathtt{M}-1]$ we have $N2_R > 0 \Rightarrow N1_R > 0$.

Arc input(s) VARIABLES1 VARIABLES2

Arc generator $PRODUCT \mapsto$ collection(variables1, variables2)

Arc arity 2

Arc constraint(s) variables1.var mod M = variables2.var mod M

Graph property(ies)
- for all connected components: **NSOURCE** ≥ **NSINK**
- **NSINK** = |VARIABLES2|

Example used_by_modulo $\left(\left\{ \begin{array}{l} \mathtt{var}-1, \\ \mathtt{var}-9, \\ \mathtt{var}-4, \\ \mathtt{var}-5, \\ \mathtt{var}-2, \\ \mathtt{var}-1 \end{array} \right\}, \left\{ \begin{array}{l} \mathtt{var}-7, \\ \mathtt{var}-1, \\ \mathtt{var}-2, \\ \mathtt{var}-5 \end{array} \right\}, 3 \right)$

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 |VARIABLES2|.

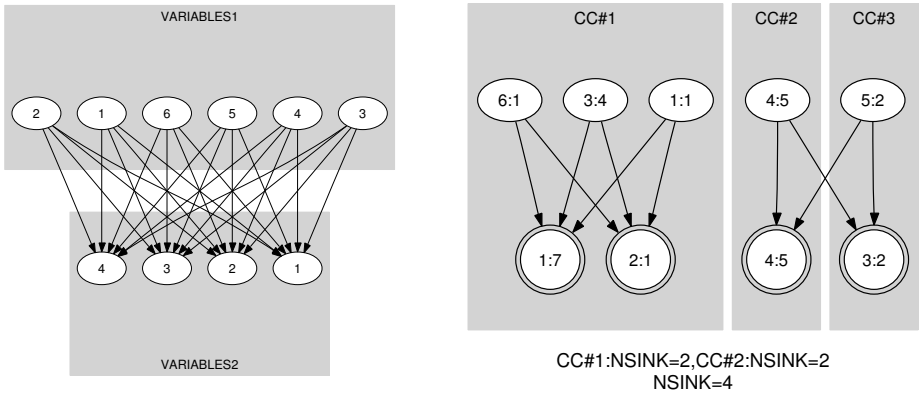


Figure 4.432: Initial and final graph of the used_by_modulo constraint

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 |VARIABLES2|. Therefore we can rewrite $\mathbf{NSINK} = |\texttt{VARIABLES2}|$ to $\mathbf{NSINK} \geq |\texttt{VARIABLES2}|$ and simplify $\underline{\overline{\mathbf{NSINK}}}$ to $\overline{\mathbf{NSINK}}$.

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)
|VARIABLES1| ≥ |VARIABLES2|
required(VARIABLES1, var)
required(VARIABLES2, var)
required(PARTITIONS, p)
|PARTITIONS| ≥ 2

Purpose For each integer i in $[1, |\texttt{PARTITIONS}|]$, 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 $[1, |\texttt{PARTITIONS}|]$ we have $N2_i > 0 \Rightarrow N1_i > 0$.

Arc input(s) VARIABLES1 VARIABLES2

Arc generator $PRODUCT \mapsto$ collection(variables1, variables2)

Arc arity 2

Arc constraint(s) in_same_partition(variables1.var, variables2.var, PARTITIONS)

Graph property(ies)
- for all connected components: $\mathbf{NSOURCE} \geq \mathbf{NSINK}$
- $\mathbf{NSINK} = |\texttt{VARIABLES2}|$

Example

$$\texttt{used\_by\_partition}\left(\left\{\begin{array}{l}\texttt{var}-1,\\ \texttt{var}-9,\\ \texttt{var}-1,\\ \texttt{var}-6,\\ \texttt{var}-2,\\ \texttt{var}-3\end{array}\right\},\left\{\begin{array}{l}\texttt{var}-1,\\ \texttt{var}-3,\\ \texttt{var}-6,\\ \texttt{var}-6\end{array}\right\},\left\{\begin{array}{l}\texttt{p}-\{\texttt{val}-1,\texttt{val}-3\},\\ \texttt{p}-\{\texttt{val}-4\},\\ \texttt{p}-\{\texttt{val}-2,\texttt{val}-6\}\end{array}\right\}\right)$$

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 |`VARIABLES2`|.

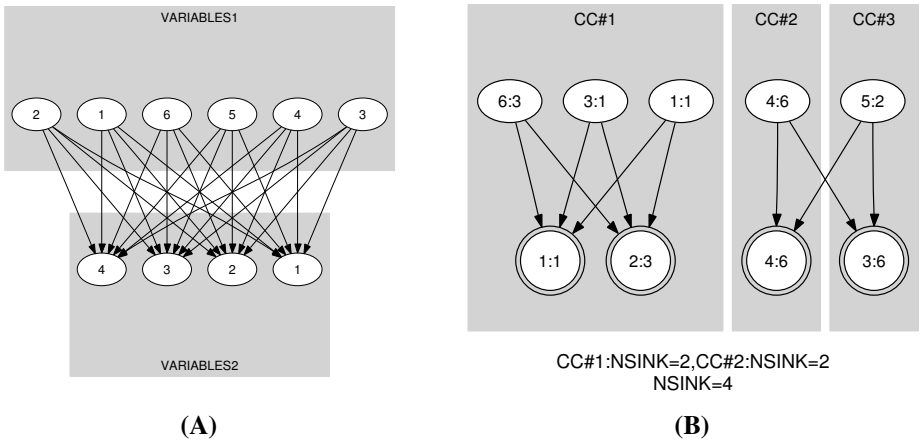


Figure 4.433: Initial and final graph of the `used_by_partition` constraint

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 |`VARIABLES2`|. Therefore we can rewrite $\mathbf{NSINK} = |\texttt{VARIABLES2}|$ to $\mathbf{NSINK} \geq |\texttt{VARIABLES2}|$ and simplify $\underline{\overline{\mathbf{NSINK}}}$ to $\overline{\mathbf{NSINK}}$.

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)

$\texttt{N} \geq 0$
$2 * \texttt{N} \leq \texttt{max}(|\texttt{VARIABLES}| - 1, 0)$
required(VARIABLES, var)

Purpose

A variable V_k $(1 < k < m)$ of the sequence of variables $\texttt{VARIABLES} = V_1, \ldots, V_m$ is a *valley* if and only if there exist an i $(1 < i \leq k)$ such that $V_{i-1} > V_i$ and $V_i = V_{i+1} = \ldots = V_k$ and $V_k < V_{k+1}$. N is the total number of valleys of the sequence of variables VARIABLES.

Example

$$\texttt{valley}\left(\begin{array}{l} 1, \left\{\begin{array}{l} \texttt{var} - 1, \\ \texttt{var} - 1, \\ \texttt{var} - 4, \\ \texttt{var} - 8, \\ \texttt{var} - 8, \\ \texttt{var} - 2, \\ \texttt{var} - 7, \\ \texttt{var} - 1 \end{array}\right\} \end{array}\right)$$

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.

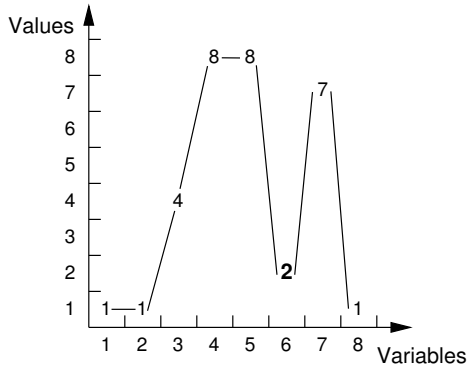


Figure 4.434: The sequence and its unique valley

Automaton Figure 4.435 depicts the automaton associated to the valley constraint. To each pair of consecutive variables $(\texttt{VAR}_i, \texttt{VAR}_{i+1})$ of the collection VARIABLES corresponds a signature variable $\texttt{S}_i$. The following signature constraint links $\texttt{VAR}_i$, $\texttt{VAR}_{i+1}$ and $\texttt{S}_i$: $(\texttt{VAR}_i < \texttt{VAR}_{i+1} \Leftrightarrow \texttt{S}_i = 0) \wedge (\texttt{VAR}_i = \texttt{VAR}_{i+1} \Leftrightarrow \texttt{S}_i = 1) \wedge (\texttt{VAR}_i > \texttt{VAR}_{i+1} \Leftrightarrow \texttt{S}_i = 2)$.

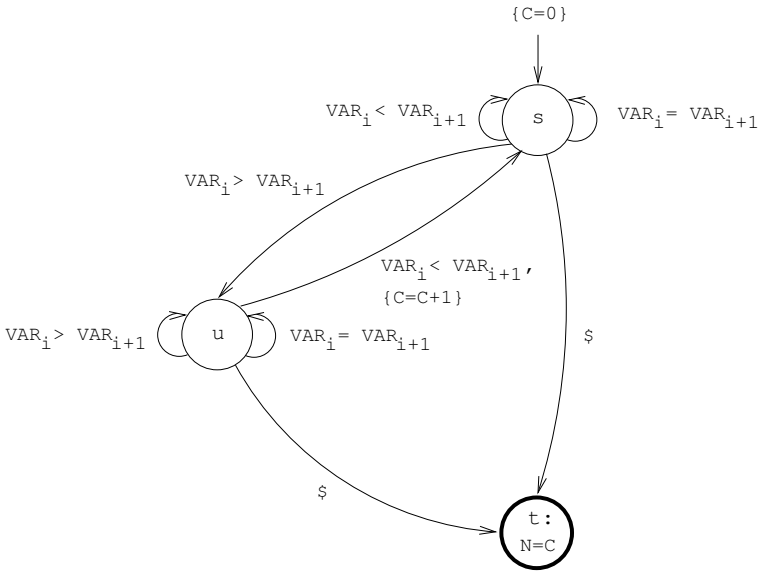


Figure 4.435: Automaton of the `valley` constraint

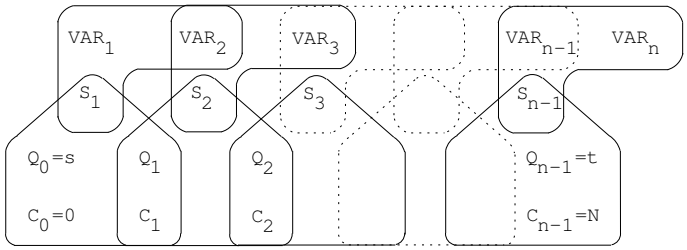


Figure 4.436: Hypergraph of the reformulation corresponding to the automaton of the `valley` constraint

Usage Useful for constraining the number of *valleys* of a sequence of domain variables.

Remark Since the arity of the arc constraint is not fixed, the `valley` 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 `no_valley`, `inflexion`, `peak`.

Key words sequence, automaton, automaton with counters, sliding cyclic(1) constraint network(2).

4.234 vec_eq_tuple

Origin Used for defining `in_relation`.

Constraint `vec_eq_tuple(VARIABLES, TUPLE)`

Argument(s)

VARIABLES	:	collection(var − dvar)
TUPLE	:	collection(val − int)

Restriction(s)
required(VARIABLES, var)
required(TUPLE, val)
|VARIABLES| = |TUPLE|

Purpose Enforce a vector of domain variables to be equal to a tuple of values.

Arc input(s) VARIABLES TUPLE

Arc generator $PRODUCT(=) \mapsto$ collection(variables, tuple)

Arc arity 2

Arc constraint(s) variables.var = tuple.val

Graph property(ies) $\mathbf{NARC} = |\text{VARIABLES}|$

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 $\mathbf{NARC}$ graph property, the arcs of the final graph are stressed in bold.

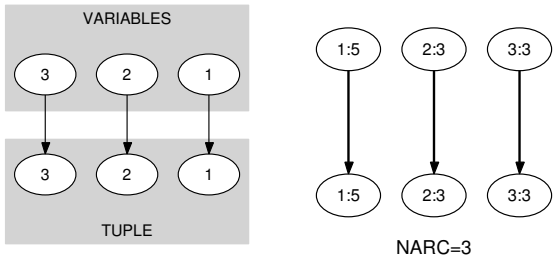


Figure 4.437: Initial and final graph of the vec_eq_tuple constraint

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 $\mathbf{NARC} = |\text{VARIABLES}|$ to $\mathbf{NARC} \geq |\text{VARIABLES}|$ and simplify $\overline{\mathbf{NARC}}$ to $\overline{\mathbf{NARC}}$.

Used in `in_relation`.

Key words value constraint, tuple.

4.235 weighted_partial_alldiff

Origin [160, page 71]

Constraint weighted_partial_alldiff(VARIABLES, UNDEFINED, VALUES, COST)

Synonym(s) weighted_partial_alldifferent, weighted_partial_alldistinct, wpa.

Argument(s)

VARIABLES	:	collection(var − dvar)
UNDEFINED	:	int
VALUES	:	collection(val − int, weight − int)
COST	:	dvar

Restriction(s)
- required(VARIABLES, var)
- required(VALUES, [val, weight])
- in_attr(VARIABLES, var, VALUES, val)
- distinct(VALUES, val)

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 explicitely defined with a weight of 0.

Arc input(s) VARIABLES VALUES

Arc generator $PRODUCT \mapsto$ collection(variables, values)

Arc arity 2

Arc constraint(s)
- variables.var $\neq$ UNDEFINED
- variables.var = values.val

Graph property(ies)
- $\mathbf{MAX\_ID} \leq 1$
- $\mathbf{SUM}$(VALUES, weight) = COST

Example

$$\texttt{weighted\_partial\_alldiff}\left(\left\{\begin{array}{l}\texttt{var}-4,\\ \texttt{var}-0,\\ \texttt{var}-1,\\ \texttt{var}-2,\\ \texttt{var}-0,\\ \texttt{var}-0\end{array}\right\},0,\left\{\begin{array}{ll}\texttt{val}-0 & \texttt{weight}-0,\\ \texttt{val}-1 & \texttt{weight}-2,\\ \texttt{val}-2 & \texttt{weight}--1,\\ \texttt{val}-4 & \texttt{weight}-7,\\ \texttt{val}-5 & \texttt{weight}--8,\\ \texttt{val}-6 & \texttt{weight}-2\end{array}\right\},8\right)$$

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`.

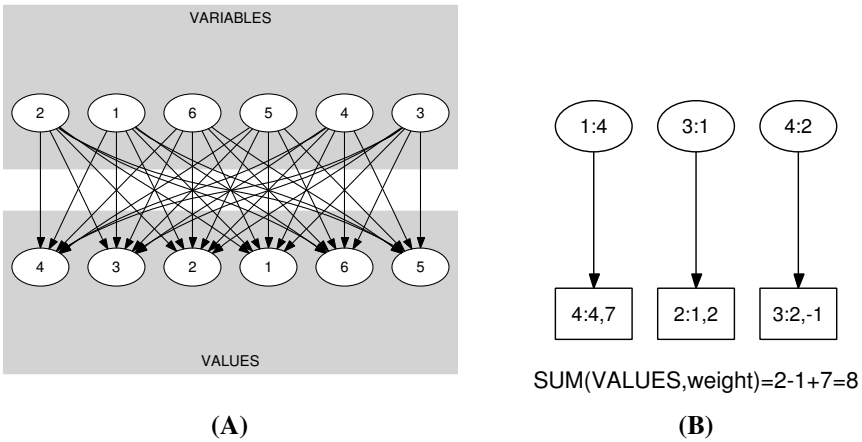


Figure 4.438: Initial and final graph of the `weighted_partial_alldiff` constraint

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 $\mathbf{MAX\_ID} \leq 1$ 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 independant 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 decribed:

- 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 :

- `Term`$_1$ `Comparison Term`$_2$ 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

```
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,1,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#<=>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-0],
[p-0],
[p-0],
[p-0],
[p-0],
[p-1],
[p-0],
[p-2],
[p-0],
[p-0],
[p-0],
[p-0],
[p-0],
[p-1],
[p-1],
[p-1],
[p-1],
[p-1],
[p-0],
[p-2],
[p-0],
[p-1],
[p-0],
[p-2],
[p-0],
[p-0],
[p-0],
[p-0],
[p-0],
[p-0],
[p-0],
[p-0],
[p-0],
[p-0],
[p-0],
[p-0],
[p-0],
[p-0],
[p-0],
[p-2],
[p-0],
[p-0],
[p-0],
[p-2],
[p-2],
[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,
    ['ORTHOTOPES'-collection(orth-'ORTHOTOPE')]).

ctr_restrictions(
    diffn,
    [size('ORTHOTOPE')>0,
     require_at_least(2,'ORTHOTOPE',[ori,siz,end]),
     'ORTHOTOPE'^siz>=0,
     required('ORTHOTOPES',orth),
     same_size('ORTHOTOPES',orth)]).

ctr_graph(
    diffn,
    ['ORTHOTOPES'],
    1,
    ['SELF'>>collection(orthotopes)],
    [orth_link_ori_siz_end(orthotopes^orth)],
    ['NARC'=size('ORTHOTOPES')]).

ctr_graph(
    diffn,
    ['ORTHOTOPES'],
    2,
    ['CLIQUE'(=\=)>>collection(orthotopes1,orthotopes2)],
    [two_orth_do_not_overlap(
         orthotopes1^orth,
         orthotopes2^orth)],
    [=('NARC',
       -(size('ORTHOTOPES')*size('ORTHOTOPES'),
         size('ORTHOTOPES')))]).

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{Carlier82}',[]).

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,1,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_greatereq

```
ctr_automaton(element_greatereq,element_greatereq).

ctr_date(element_greatereq,['20030820','20040530']).

ctr_origin(
    element_greatereq,
    '\\cite{OttossonThorsteinssonHooker99}',
    []).

ctr_arguments(
    element_greatereq,
    ['ITEM'-collection(index-dvar,value-dvar),
     'TABLE'-collection(index-int,value-int)]).

ctr_restrictions(
    element_greatereq,
    [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_greatereq,
    ['ITEM','TABLE'],
    2,
    ['PRODUCT'>>collection(item,table)],
    [item^index=table^index,item^value>=table^value],
    ['NARC'=1]).

ctr_example(
    element_greatereq,
    element_greatereq(
        [[index-1,value-8]],
        [[index-1,value-6],
         [index-2,value-9],
         [index-3,value-2],
         [index-4,value-9]])).

element_greatereq(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 heighest_peak

```
ctr_automaton(heighest_peak,heighest_peak).

ctr_date(heighest_peak,['20040530']).

ctr_origin(heighest_peak,'Derived from %c.',[peak]).

ctr_arguments(
    heighest_peak,
    ['HEIGHT'-dvar,'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    heighest_peak,
    ['HEIGHT'>=0,'VARIABLES'^var>=0,required('VARIABLES',var)]).

ctr_example(
    heighest_peak,
    heighest_peak(
        8,
        [[var-1],
         [var-1],
         [var-4],
         [var-8],
         [var-6],
         [var-2],
         [var-7],
         [var-1]])).

heighest_peak(A,B) :-
        heighest_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]).

heighest_peak_signature([],[],[]).

heighest_peak_signature([A],[],[]).

heighest_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,
        heighest_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 explicitely 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{BourdaisGalinierPesant03}',[]).

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-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,
    ['ORTHOTOPE'-collection(ori-dvar,siz-dvar,end-dvar)]).

ctr_arguments(
    two_orth_column,
    ['ORTHOTOPE1'-'ORTHOTOPE',
     'ORTHOTOPE2'-'ORTHOTOPE',
     'N'-int]).

ctr_restrictions(
    two_orth_column,
    [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_column,
    ['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),
              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 Vizualisation 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 Hoeve, 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. Régin 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. Régin. 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, Compiégne, 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. 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 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 csps: 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] Tallys 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