



HAL
open science

Check-in Desk Scheduling Optimisation at CDG International Airport

Thibault Falque, Gilles Audemard, Christophe Lecoutre, Bertrand Mazure

► **To cite this version:**

Thibault Falque, Gilles Audemard, Christophe Lecoutre, Bertrand Mazure. Check-in Desk Scheduling Optimisation at CDG International Airport. Doctoral Program of the 29th International Conference on Principles and Practice of Constraint Programming (CP 2023), Aug 2023, Toronto, Canada. hal-04250561

HAL Id: hal-04250561


<https://hal.science/hal-04250561v1>

Submitted on 19 Oct 2023

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

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

1 Check-in Desk Scheduling Optimisation at CDG 2 International Airport

3 **Thibault Falque** ✉ 

4 Exakis Nelite

5 CRIL, Univ Artois & CNRS

6 **Gilles Audemard** ✉ 

7 CRIL, Univ Artois & CNRS

8 **Christophe Lecoutre** ✉ 

9 CRIL, Univ Artois & CNRS

10 **Bertrand Mazure** ✉ 

11 CRIL, Univ Artois & CNRS

12 — Abstract —

13 More than ever, air transport players (i.e., airline and airport companies) in an intensely competitive
14 climate need to benefit from a carefully optimized management of airport resources to improve the
15 quality of service and control the induced costs. In this paper, we investigate the Airport Check-in
16 Desk Assignment Problem. We propose a Constraint Programming (CP) model for this problem,
17 and present some promising experimental results from data coming from ADP (Aéroport de Paris).

18 **2012 ACM Subject Classification** Artificial intelligence

19 **Keywords and phrases** constraint optimization problem, modeling, application, Paris airport

20 **Digital Object Identifier** 10.4230/LIPIcs.DPCP.2023.

21 **1** Introduction

22 Before the COVID-19 health crisis, the International Air Transport Association (IATA)
23 forecasts showed that passengers would double by 2036, reaching 7.8 billion. The COVID-19
24 pandemic has slowed air traffic considerably, especially in 2020 and early 2021, but since then
25 the economic pressure is back again. Actually, air traffic picked up in 2022 and is similar
26 to 2019. Some airlines have even announced the return to service of Airbus 380 to manage
27 demand. In such a context, optimizing airport resources management remains essential
28 to control induced costs while keeping a good quality of services. For many planning and
29 scheduling air transport problems, techniques and tools developed from mathematical and
30 constraint programming remain essential. Specifically, when airline companies have access
31 to the resources delivered at the airport, the consumption of these resources (e.g., check-in
32 banks, aircraft stand) must be carefully planned while optimizing an objective function
33 determined by some business rules; see, for example, [24, 9, 22, 8, 29].

34 A classical air transport problem is the Airport Gate Assignment Problem (AGAP),
35 which involves assigning each flight (aircraft) to an available gate while maximizing both
36 passenger conveniences and the airport's operational efficiency; see surveys in [2, 5] and
37 models in [19, 20, 23]. Another classical problem is the Check-in Assignment Problem,
38 which involves assigning each flight to one or more check-in desks depending on the airline's
39 requirements. Different approaches in MILP (mixed-integer linear programming) have been
40 proposed [33, 1]. A recent survey [13] presents different methods for solving this problem using
41 integer programming or dynamic programming. However, it does not seem to indicate that
42 constraint programming modeling is proposed. Because significant improvements have been
43 made during the last decade in Constraint Programming (CP), such as, e.g., efficient filtering



© Thibault Falque, Gilles Audemard, Christophe Lecoutre and Bertrand Mazure;
licensed under Creative Commons License CC-BY 4.0

Doctoral Program of the 29th International Conference on Principles and Practice of Constraint Programming.

Editor: Xavier Gillard; Article No. ; pp. :1–:9

Leibniz International Proceedings in Informatics



LIPIC Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

XX:2 Check-in Desk Scheduling Optimisation at Paris Airport

44 (and compression) algorithms for table constraints [14, 6], or lazy clause generation [12],
45 tackling optimization of airport tasks with CP remains an interesting issue. In this paper,
46 we are interested in the Airport Check-In Desk Assignment Problem as defined at CDG
47 International Airport. We propose a Constraint Programming (CP) approach and show its
48 potential interest by presenting a few promising experimental results.

49 The rest of this paper is organized as follows. In Section 2, we present Airport Check-In
50 Desk Assignment Problem. In Section 3, we propose a Constraint Optimization model
51 for this problem and some possible variants of this model. In Section 4, we present some
52 experiments carried out in an in-situ experimental context with the Paris airport system.
53 Finally, in Section 5, we conclude and give some perspectives for future works.

54 2 Airport Check-In Desk Assignment Problem at Paris Airport

55 CDG Airport is the ninth-largest airport in the world in terms of passenger traffic. There
56 are approximately 1,400 flight movements (takeoff or landing) per day. At the airport, one
57 of the combinatorial problems to address is to set each flight (or group of flights) to one
58 or more available check-in desks. In this section, we provide some information about the
59 Airport Check-In Desk Assignment Problem.

60 A *registration* corresponds to a flight or a set of flights of the same airline. For each
61 registration, a task must be carried out: associating a set of check-in desks with it. Each
62 *task* of registration (or check-in for a flight) starts at the same time and ends at the same
63 time. Note that the number of check-in desks depends on the number of passengers and is
64 fixed in advance by the airline and the airport. Figure 1 presents some registration tasks at
65 Orly Airport with 1, 4 and 5 tasks.

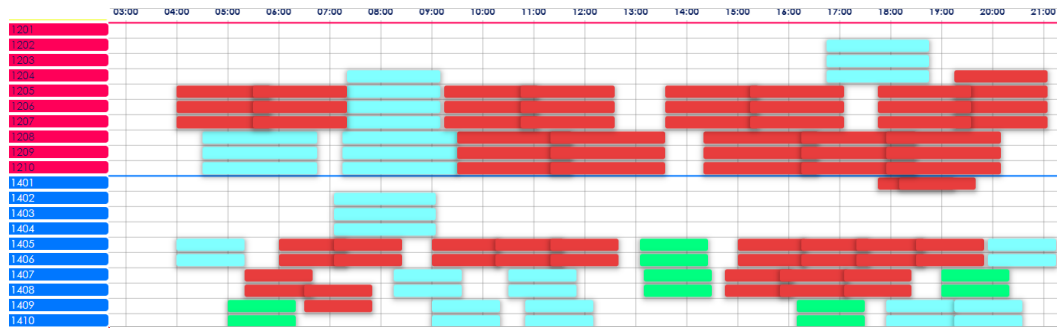


■ **Figure 1** An example of planning at Orly Airport

66 Planning registrations can be achieved for one or more days. For the moment, the
67 planning horizon we manage is for one week (sometimes less). In the rest of the paper, a
68 check-in desk will be called a bank, and the set of all registrations (tasks) is denoted by
69 \mathcal{R} , the set of all zones (groups of banks) is denoted by \mathcal{Z} , the set of banks by \mathcal{C} , and the
70 maximal number of banks required by a registration by ν .

71 2.1 Imposing Consecutive Desks

72 When attempting to model this problem, a first arising constraint is that the banks (check-in
73 desks) used for a specific registration must be **consecutive** (as we can observe in Figure 1).
74 Importantly, as banks are grouped by zones, we must pay attention to assign only banks
75 from the same zone to a registration. For example, in Figure 2, there are two zones (colored
76 in blue and pink); so for a registration, we cannot use both a blue and a pink bank.



■ **Figure 2** Example of a planning that allows overlapping.

77 2.2 Sharing Desks under Conditions

78 By default, a registration cannot share its assigned banks with those of another registration
 79 if the two registration tasks are time overlapping. So at any time, no bank can be shared by
 80 two different registrations. However, for some reason of logistics (space) and under certain
 81 general conditions (called overlapping rules), some overlapping between flights from the same
 82 airline company may be tolerated for a limited period of time and or for a limited number of
 83 tasks. In the latter case, if for example the number of banks required by a registration is
 84 set to 4 and the maximum number of overlapping situations is 2, then only two banks from
 85 the four banks associated with the registration can be shared with another registration that
 86 shares the same overlapping rule. We will note \mathcal{O} the set of pairs of registrations (ρ_1, ρ_2)
 87 that can't strictly share banks (they may be time overlapping, but no rule exists permitting
 88 to have shared banks between them). Figure 2 presents an example of planning that allows
 89 overlapping for 100% of the time and without a limited number of tasks.

90 2.3 Excluding some Banks

91 It is frequent that some banks are unavailable for a period of several hours to several days
 92 (for example for maintenance reasons). Also, some *exclusion constraints* ensures that certain
 93 banks are excluded from certain registrations under some conditions.

94 2.4 Pre-assigning Banks

95 Sometimes, users (from ADP) may want to force a specific set of banks to be associated with
 96 some registrations. We will note (ρ, j, c) the triplet that represents the pre-assignment of
 97 bank c as the j th bank used by registration ρ ; all such triplets will be denoted by \mathcal{P} .

98 2.5 Specifying the Objective

99 Of course, assigning a bank to a registration is subject to some placement preferences by
 100 airline companies. For each assigned bank, a reward is given: the reward of assigning the
 101 bank c as the j th bank used by registration ρ is denoted by $r_{\rho,j}^c$. Assuming that we have a
 102 series¹ of 0/1 variables $x_{\rho,j}^c$ associated with each registration task (indicating which check-in
 103 desk will be used), we can then define the overall objective function as follows:

¹ Note that we shall not use 0/1 variables in the model proposed in Section 3.

$$104 \quad \text{maximize} \quad \sum_{\substack{\rho \in \mathcal{R} \\ j \in 1..ntasks(\rho)}} x_{\rho,j}^c \times r_{\rho,j}^c \quad (1)$$

105 **3** Constraint Optimization Model

106 Now that the problem has been introduced in general terms, we need to describe it more
 107 formally using a constraint network. A *Constraint Network* (CN) consists of a finite set
 108 of variables subject to a finite set of constraints. Each variable x can take a value from a
 109 finite set called the *domain* of x . Each constraint c is specified by a relation that is defined
 110 over (the Cartesian product of the domains of) a set of variables. A *solution* of a CN is the
 111 assignment of a value to every variable such that all constraints are satisfied. A *Constraint*
 112 *Network under Optimization* (CNO) is a constraint network that additionally includes an
 113 objective function `obj` that maps any solution to a value in \mathbb{R} .

114 For modeling CNOs, also called Constraint Optimization Problems (COPs), several
 115 modeling languages or libraries exist such as, e.g., OPL [27], MiniZinc [25, 30], Essence [11]
 116 and PyCSP³ [18]. Our choice is the recently developed Python library PyCSP³ that permits
 117 to generate specific instances (after providing ad hoc data) in XCSP3 format [3, 4], which is
 118 recognized by some well-known CP solvers such as ACE (AbsCon Essence) [16], OsaR [26],
 119 Choco [28], and PicatSAT [34]. For simplicity, however, we formally describe below the
 120 model developed for the Airport Check-in desk problem in a higher “mathematical” form.

121 Firstly, we need to introduce the variables of our model. A registration must use check-in
 122 desks in coherence with its strategy. Therefore, rather than making domains containing all
 123 possible banks, the domains are initially reduced to the banks that are compatible with the
 124 strategy associated with the registration. For each registration ρ we note this domain $\mathcal{D}_{x,\rho}$.
 125 Similarly, the domains for the variables representing rewards for airlines, they only contain
 126 the values corresponding to the allowed check-in desks. For each registration ρ we note this
 127 domain $\mathcal{D}_{r,\rho}$. We also introduce a fictive bank f with a reward of 0.

128 We need two (2-dimensional) arrays of variables to represent assigned registration and
 129 associated rewards:

- 130 ■ x is a matrix of $|\mathcal{R}| \times \nu$ variables having the set of values $\mathcal{D}_{x,\rho}$; $x[\rho][j]$ represents the
 131 index (code) of the check-in desk assigned to the j th task of the registration ρ .
- 132 ■ r is a matrix of $|\mathcal{R}| \times \nu$ variables having the set of values $\mathcal{D}_{r,\rho}$; $r[\rho][j]$ represents the
 133 satisfaction of the airline for the j th task of the registration ρ .

134 Secondly, we need to introduce the constraints in our model. Because of the nature of
 135 the problem (and data), it is natural to post so-called table constraints, which explicitly
 136 enumerate either the allowed tuples (positive table) or the disallowed tuples (negative table)
 137 for a sequence of variables (representing the scope of a constraint). Efficient algorithms for
 138 such table constraints have been developed over the last decade [15, 17, 6, 31].

139 **3.1** A first COP formulation

140 Let us consider the variables previously introduced, the problem can be formulated as follows:

$$141 \quad x[\rho][j] = c, \forall (\rho, j, c) \in \mathcal{P} \quad (2)$$

$$142 \quad (x[\rho][j] = x[\rho][j+1] - 1) \vee (x[\rho][j] = f \wedge x[\rho][j+1] = f), \forall \rho \in \mathcal{R}, \forall j \in \text{ntask}(\rho) \quad (3)$$

$$143 \quad x[\rho_1][i] \neq x[\rho_2][j] \forall \rho_1, \rho_2 \in \mathcal{O}, \forall i \in \text{ntask}(\rho_1), \forall j \in \text{ntask}(\rho_2) \quad (4)$$

$$144 \quad x[\rho][i], r[\rho][i] \in \{(c, w_\rho^c), \forall c \in \mathcal{C} \cup \{f, 0\}\} \quad (5)$$

145 Constraints (2) ensure that each pre-assignment of \mathcal{P} is respected. Constraints (3) ensure
 146 that the chosen check-in desks for registration are consecutive or used the the fictive check-in
 147 desk for each task of each registration. The introduction of holes in the domains (e.g., useless
 148 check-in desks) makes it possible to manage this by imposing that a task must be equal
 149 to the following task minus one and by not including useless check-in desks in the domain.
 150 In this way, we insert a hole representing the zone's separation. Constraints (4) prevent
 151 two overlapping registration from being assigned to the same check-in desk (as presented in
 152 Section 2.2). Constraints (5) use table constraint to map the check-in desk with this weight.
 153 We note w_ρ^c the weight associated to the check-in desk c considering the strategy rule of ρ .
 154 Finally, for constraints presented in Section 2.3 we use conflict tables.

155 3.2 Gathering Binary Difference Constraints

156 We will now strengthen this natural formulation by reformulating the set of constraints (4)
 157 using the *AllDifferentExcept* constraint. This latter enforces all variables to take distinct
 158 values, except those variables that are assigned value to a special (joker) value (here it is our
 159 fictive bank f).

$$160 \quad \text{AllDifferentExcept}(\{x[\rho_1], x[\rho_2]\}, f) \quad (6)$$

161 For each pair ρ_1, ρ_2 in the set of forbidden overlaps \mathcal{O} . Note that we used the notation
 162 $x[\rho_1]$ and $x[\rho_2]$ for a shortcut that integrates the entire second dimension of the matrix into
 163 the constraint (i.e. each task of ρ_1 or ρ_2). This formulation allows us to reduce the number
 164 of constraints about no-overlapping tasks considerably, as the previous formulation needs a
 165 quadratic number of *not-equal* constraints.

166 3.3 Gathering *AllDifferentExcept* constraints

167 Even though the formulation above notably reduces the number of posted constraints, the
 168 solver remains too slow for finding acceptable results (bounds) in a reasonable amount of
 169 time. We have thus gathered all *AllDifferentExcept* constraints into a unique pragmatic
 170 constraint called *GatherAllDifferentExcept*. For this particular constraint, we use a
 171 specific fast propagator that performs a limited form of filtering (i.e., does not enforce
 172 generalized arc consistency). This is a very pragmatic approach, which is somewhat equivalent
 173 to the initial set of binary constraints, but quite faster (only one constraint being posted).

■ **Table 1** Description of the main instances (Φ) uses in this study.

Instance	$ \mathcal{C} $	# Tasks	$ \mathcal{O} $	# Strategy rules
ORLY 1234 - 2023-05-08/2023-05-14	326	2224	21	31
CDG T1 - 2023-07-03/2023-07-09	137	630	6	38
CDG T2 - B & D - 2023-07-03/2023-07-09	108	766	1	21

174 4 Experiments Results

175 4.1 Instances

176 Table 1 presents some factual aspects concerning our instances based on real data from Paris
 177 Airport and representing realistic scenarios. The first column is the name of the instance,
 178 while the second, third, fourth and fifth columns indicate the number of check-in desks, the
 179 number of tasks, the number of overlapping rules and strategy rules, respectively. Each
 180 instance concerns a planning over 7 consecutive days. We note this set of instances Φ .

181 4.2 Decomposition

182 Because decomposing the problem is possible without degrading results, it was decided to
 183 break the problem into simpler sub-problems so as to solve them successfully. To do this,
 184 we first break down the problem into groups of terminals based on assignment strategies.
 185 If a strategy for registration covers the check-in desks of several terminals then we group
 186 the terminals, otherwise, we leave them separate. Finally, for each group of terminals, we
 187 can re-decompose them day by day. If there are night flights, these are pre-assigned before
 188 launching resolution.

189 4.3 Results

190 In our experimentation, the time limit for each execution (part of the decomposition) is
 191 limited to 30 seconds (compilation time in XCSP format is not included in this timeout).
 192 They are launched on a real environment in the Paris airport system equipped with 64
 193 GB of RAM and two 10-core Intel Xeon Silver 4210R (2.4 GHZ). Note that the solver is
 194 stopped when no more improvement has been made during a period of 5 seconds (since the
 195 last solution was found). Since the choice of stopping the solver after 5 seconds makes it
 196 non-deterministic, we run each configuration on each decomposition 5 times. For our study,
 197 we use `frba/dom` [21] as variable-ordering heuristic (this heuristic was observed as the best
 198 one on this problem), `solution-saving` [32, 7] for simulating a form of large neighborhood
 199 search. Concerning the value-ordering heuristic we have tested different configurations:
 200 `BIVS` [10], until the first solution is found (after that, the smallest value in the domain is
 201 systematically selected if solution-saving cannot be applied), `Static`, a static order based
 202 on the rewards of check-in desks in the strategies, and `BIVS + Static`, an approach that
 203 mixes the two heuristics: `BIVS` until the first solution is found (after that, `Static` is used if
 204 solution-saving cannot be applied). We use the solver `ACE`² and in particular the `JUniverse`³

² <https://github.com/xcsp3team/ace>

³ <https://github.com/crillab/juniverse>

■ **Table 2** Result on decomposition ORY124 (7 days) from planning ORLY 1234 of Φ .

Configuration	ALE	GAT	Time	Solver T	First	Last	# not
Bivs + Static	0	False	140-201	101-146	22,648,100 (35-70)	22,834,100 (48-90)	281
Bivs + Static	0	True	148-186	88-120	22,648,100 (5-10)	23,676,100 (31-49)	256
Bivs + Static	4	False	173-222	103-144	22,648,100 (37-53)	22,840,100 (50-68)	281
Bivs + Static	4	True	139-182	80-102	22,648,100 (5-10)	23,694,100 (30-50)	255
Bivs	0	False	141-209	105-138	22,648,100 (37-63)	22,840,100 (51-81)	281
Bivs	0	True	118-159	77-110	22,648,100 (5-7)	23,620,900 (26-44)	258
Bivs	4	False	182-228	106-155	22,648,100 (36-61)	22,840,100 (49-78)	281
Bivs	4	True	145-188	78-105	22,648,100 (5-11)	23,565,100 (23-47)	260
Static	0	False	235-324	194-240	17,512,500 (16-25)	19,762,600 (165-210)	288
Static	0	True	119-187	80-112	17,512,500 (4-7)	23,940,100 (30-44)	243
Static	4	False	250-300	196-230	17,512,500 (17-30)	19,466,400 (155-201)	288
Static	4	True	112-177	77-104	17,512,500 (4-6)	23,782,700 (25-42)	245

■ **Table 3** Result on decomposition ORY3 (7 days) from planning ORLY 1234 of Φ .

Configuration	ALE	GAT	Time	Solver T	First	Last	# not
Bivs + Static	0	False	287-328	208-241	19,894,700 (132-194)	19,896,100 (152-204)	84
Bivs + Static	0	True	186-245	100-150	21,082,800 (6-18)	21,351,080 (20-69)	12
Bivs + Static	4	False	271-362	184-254	21,082,800 (103-162)	21,085,540 (120-191)	16
Bivs + Static	4	True	163-235	82-122	21,082,800 (5-11)	21,392,580 (22-65)	12
Bivs	0	False	277-338	178-231	21,082,800 (101-165)	21,085,860 (130-188)	16
Bivs	0	True	178-275	80-152	21,082,800 (6-14)	21,342,520 (17-77)	13
Bivs	4	False	266-359	172-256	21,082,800 (99-153)	21,085,040 (117-176)	16
Bivs	4	True	179-267	98-162	21,082,800 (4-13)	21,371,480 (22-62)	12
Static	0	False	281-381	190-285	18,376,800 (16-37)	19,235,000 (148-209)	15
Static	0	True	253-334	173-240	18,376,800 (4-11)	21,469,680 (108-203)	1
Static	4	False	298-344	220-256	18,376,800 (20-47)	19,215,220 (183-209)	14
Static	4	True	276-348	199-246	18,376,800 (4-12)	21,569,760 (151-206)	1

205 adapter of ACE: ACEURANCETOURIX⁴ which allows interaction with ACE via an API.

206 Table 2 and 3 present some results with different configurations for instance ORLY 1234
 207 for the week from 2023-05-08 to 2023-05-14. After the decomposition step, this latter is
 208 decomposed in two groups of terminals (*ORLY 1,2,4* and *ORLY 3*). The first column
 209 presents the configuration of ACE, and the second column indicates the arity limit for which
 210 intensification constraints are transformed into extension constraints by the solver. The third
 211 column indicates if we gather or not the `AllDifferentExcept` constraints, *False* corresponds
 212 to the second formulation (Section 3.2) and *True* to the third formulation (Section 3.3). The
 213 column *Time* presents the best and worst case of resolution time (including compilation time)
 214 over the 5 executions if we have run the resolution sequentially. The column *Solver T* is
 215 similar to the column *Time* but only for the solver. The columns *First* and *Last* contain the
 216 mean of the first (resp. last) bound computed over the 5 executions and the best and worst
 217 case runtime for obtaining the first (resp. last) bound. Finally, the last column contains the
 218 number of registrations that are not assigned (i.e. the number of registrations that use the
 219 fictive check-in desk). For space reasons, we have limited the results to only ORLY instance
 220 but all the results are available and reproducible⁵ (thanks to *Metrics*⁶).

221 We can see that the value-heuristic `Static` finds a good solution and reduces the number
 222 of registrations that are not assigned. However, the time needed to find this solution is higher

⁴ <https://github.com/crillab/aceurancetourix>

⁵ <https://gitlab.com/productions-tfalque/articles/check-in-scheduling-optim-cdg-airport/experiments>

⁶ <https://github.com/crillab/metrics>

223 than BIVS, which finds an acceptable solution. The combination of both does not allow to
 224 reduce time or reduce the number of unassigned registrations. We can also observe that in
 225 the case of the `Static` configuration, `GatherAllDifferent` brings a 22% gain and reduces
 226 unallocated registrations by 45 in an acceptable amount of time.

227 **5 Conclusion**

228 In this paper, we have been interested in the Airport Check-in Desk Problem as defined at
 229 Paris Airport. We have proposed a COP model for this problem, mainly exploiting table
 230 constraints and developing an ad-hoc constraint (for reducing the number of constraints, and
 231 accelerating the resolution consequently). We have presented a first empirical evaluation of
 232 our approach. Our results look quite promising, as the ADP group starts replacing their
 233 current proprietary solution with ours, based on generic open-source tools (modeling library
 234 and constraint solver). In the future, we plan to simplify the execution process by limiting
 235 data processing in the PyCSP³ model and discarding the decomposition step. We also plan
 236 to use a multi-criteria objective.

237 **References**

-
- 238 **1** G. E Araujo and H. M Repolho. Optimizing the Airport Check-In Counter Allocation Problem.
 239 *Journal of Transport Literature*, 9(4):15–19, December 2015. doi:10.1590/2238-1031.jtl.
 240 v9n4a3.
 - 241 **2** A Bouras, M. A Ghaleb, U. S Suryahatmaja, and A. M Salem. The Airport Gate Assignment
 242 Problem: A Survey. *The Scientific World Journal*, 2014:e923859, November 2014. doi:
 243 10.1155/2014/923859.
 - 244 **3** F. Boussemart, C. Lecoutre, G. Audemard, and C. Piette. XCSP3: An integrated format for
 245 benchmarking combinatorial constrained problems. *CoRR*, abs/1611.03398, 2016.
 - 246 **4** F. Boussemart, C. Lecoutre, G. Audemard, and C. Piette. XCSP3-core: A format for
 247 representing constraint Satisfaction/Optimization problems. *CoRR*, abs/2009.00514, 2020.
 248 arXiv:2009.00514.
 - 249 **5** G. S Dağ, F Gzara, and T Stützle. A review on airport gate assignment problems: Single
 250 versus multi objective approaches. *Omega*, 92:102146, April 2020. doi:10.1016/j.omega.
 251 2019.102146.
 - 252 **6** J. Demeulenaere, R. Hartert, C. Lecoutre, G. Perez, L. Perron, J.-C. Régim, and P. Schaus.
 253 Compact-Table: Efficiently Filtering Table Constraints with Reversible Sparse Bit-Sets. In
 254 *Proceedings of CP'16*, pages 207–223, 2016.
 - 255 **7** E. Demirovic, G. Chu, and P. Stuckey. Solution-based phase saving for CP: A value-selection
 256 heuristic to simulate local search behavior in complete solvers. In *Proceedings of CP'18*, pages
 257 99–108, 2018.
 - 258 **8** G Diepen, J.M. Akker, J.A. Hoogeveen, and J.W. Smeltink. Using column generation for gate
 259 planning at Amsterdam Airport Schiphol. January 2007.
 - 260 **9** M Dincbas and H Simonis. APACHE - A constraint based, automated stand allocation system.
 261 Automated Stand Allocation System Proc. Of Advanced Software Technology in Air Transport
 262 (ASTAIR'91) Royal Aeronautical Society, pages 267–282, October 1991.
 - 263 **10** J.-G Fages and C Prud'Homme. Making the First Solution Good! In *ICTAI 2017*, pages
 264 1073–1077, Boston, MA, November 2017. IEEE. doi:10.1109/ICTAI.2017.00164.
 - 265 **11** A. Frisch, M. Grum, C. Jefferson, B. M Hernandez, and I. Miguel. The design of ESSENCE: A
 266 constraint language for specifying combinatorial problems. In *Proceedings of IJCAI'07*, pages
 267 80–87, 2007.
 - 268 **12** G. Chu, P. Stuckey, M. Garcia de la Banda, and C. Mears. Symmetries and Lazy Clause
 269 Generation. In *Proceedings of IJCAI'11*, pages 516–521, 2011.

- 270 13 T. R. Lalita and G. S. R. Murthy. The Airport Check-in Counter Allocation Problem: A
271 Survey, August 2022. [arXiv:2208.13544](#).
- 272 14 B. Le Charlier, M. T. Khong, C. Lecoutre, and Y. Deville. Automatic Synthesis of Smart
273 Table Constraints by Abstraction of Table Constraints. In *Proceedings of IJCAI'17*, pages
274 681–687, 2017.
- 275 15 C Lecoutre. STR2: Optimized Simple Tabular Reduction for Table Constraints. *Constraints :
276 an international journal*, 16(4):341–371, 2011.
- 277 16 C. Lecoutre. ACE, a generic constraint solver. *CoRR*, abs/2302.05405, 2023. [arXiv:2302.
278 05405](#), doi:10.48550/arXiv.2302.05405.
- 279 17 C. Lecoutre, C. Likitvivanavong, and R. Yap. STR3: A path-optimal filtering algorithm for
280 table constraints. *Artificial Intelligence*, 220:1–27, 2015.
- 281 18 C. Lecoutre and N. Szczepanski. PyCSP3: Modeling combinatorial constrained problems in
282 Python. *CoRR*, abs/2009.00326, 2020. [arXiv:2009.00326](#).
- 283 19 C Li. Airport Gate Assignment: New Model and Implementation. *arXiv:0811.1618 [cs]*,
284 November 2008. [arXiv:0811.1618](#).
- 285 20 C Li. Airport Gate Assignment A Hybrid Model and Implementation. *arXiv:0903.2528 [cs]*,
286 March 2009. [arXiv:0903.2528](#).
- 287 21 H Li, M Yin, and Z Li. Failure Based Variable Ordering Heuristics for Solving CSPs. In L. D
288 Michel, editor, *CP '21*, volume 210, pages 9:1–9:10, 2021. doi:10.4230/LIPICs.CP.2021.9.
- 289 22 A. Lim, B. Rodrigues, and Y. Zhu. Airport Gate Scheduling with Time Windows. *Artificial
290 intelligence review*, 24(1):5–31, September 2005. doi:10.1007/s10462-004-7190-4.
- 291 23 J. L’Ortye, M. Mitici, and H.G. Visser. Robust flight-to-gate assignment with landside
292 capacity constraints. *Transportation Planning and Technology*, 44(4):356–377, May 2021.
293 doi:10.1080/03081060.2021.1919347.
- 294 24 R. S. Mangoubi and D. F. X. Mathaisel. Optimizing Gate Assignments at Airport Terminals.
295 *Transportation Science*, 19(2):173–188, 1985.
- 296 25 N. Nethercote, P. Stuckey, R. Becket, S. Brand, G. Duck, and G. Tack. MiniZinc: Towards a
297 Standard CP Modelling Language. In *Proceedings of CP'07*, pages 529–543, 2007.
- 298 26 OcaR Team. OcaR: Scala in OR, 2012.
- 299 27 P. van Hentenryck. *The OPL Optimization Programming Language*. The MIT Press, 1999.
- 300 28 C. Prud’homme, J.-G. Fages, and X. Lorca. Choco-solver, TASC, INRIA Rennes, LINA,
301 Cosling S.A. 2016.
- 302 29 H Simonis. Models for global constraint applications. *Constraints : an international journal*,
303 12(1):63–92, March 2007. doi:10.1007/s10601-006-9011-7.
- 304 30 P. Stuckey, R. Becket, and J. Fischer. Philosophy of the MiniZinc challenge. *Constraints*,
305 15(3):307–316, 2010.
- 306 31 H. Verhaeghe, C. Lecoutre, and P. Schaus. Extending Compact-Table to Negative and Short
307 Tables. In *Proceedings of AAAI'17*, pages 3951–3957, 2017.
- 308 32 J. Vion and S. Piechowiak. Une simple heuristique pour rapprocher DFS et LNS pour les
309 COP. In *Proceedings of JFPC'17*, pages 39–45, 2017.
- 310 33 S Yan, C.-H Tang, and M Chen. A model and a solution algorithm for airport common
311 use check-in counter assignments. *Transportation Research Part A: Policy and Practice*,
312 38(2):101–125, February 2004. doi:10.1016/j.tra.2003.10.001.
- 313 34 N. F. Zhou, H. Kjellerstrand, and J. Fruhman. *Constraint Solving and Planning with Picat*.
314 Springer, 2017.