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, Aug 2023, Toronto, Canada. hal-04250561

HAL Id: hal-04250561
https://hal.science/hal-04250561
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.
Check-in Desk Scheduling Optimisation at CDG International Airport

Thibault Falque \envelope
Exakis Nelite
CRIL, Univ Artois & CNRS
Gilles Audemard \envelope
CRIL, Univ Artois & CNRS
Christophe Lecoutre \envelope
CRIL, Univ Artois & CNRS
Bertrand Mazure \envelope
CRIL, Univ Artois & CNRS

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

2012 ACM Subject Classification Artificial intelligence

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


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

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

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

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

2 Airport Check-In Desk Assignment Problem at Paris Airport

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

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

![Figure 1 An example of planning at Orly Airport](image)

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

2.1 Imposing Consecutive Desks

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

By default, a registration cannot share its assigned banks with those of another registration if the two registration tasks are time overlapping. So at any time, no bank can be shared by two different registrations. However, for some reason of logistics (space) and under certain general conditions (called overlapping rules), some overlapping between flights from the same airline company may be tolerated for a limited period of time and or for a limited number of tasks. In the latter case, if for example the number of banks required by a registration is set to 4 and the maximum number of overlapping situations is 2, then only two banks from the four banks associated with the registration can be shared with another registration that shares the same overlapping rule. We will note $O$ the set of pairs of registrations $(\rho_1, \rho_2)$ that can’t strictly share banks (they may be time overlapping, but no rule exists permitting to have shared banks between them). Figure 2 presents an example of planning that allows overlapping for 100% of the time and without a limited number of tasks.

2.3 Excluding some Banks

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

2.4 Pre-assigning Banks

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

2.5 Specifying the Objective

Of course, assigning a bank to a registration is subject to some placement preferences by airline companies. For each assigned bank, a reward is given: the reward of assigning the bank $c$ as the $j$th bank used by registration $\rho$ is denoted by $r_{\rho,j}^c$. Assuming that we have a series\(^1\) of 0/1 variables $x_{\rho,j}^c$ associated with each registration task (indicating which check-in 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.
maximize \sum_{\rho \in \mathcal{R}} x_{\rho,j}^c \times r_{\rho,j}^c \quad (1)

3 Constraint Optimization Model

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

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

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

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

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

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

3.1 A first COP formulation

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

\[ x[\rho][j] = c, \forall (\rho, j, c) \in \mathcal{P} \quad (2) \]
Constraints (2) ensure that each pre-assignment of \( P \) is respected. Constraints (3) ensure that the chosen check-in desks for registration are consecutive or used the fictive check-in desk for each task of each registration. The introduction of holes in the domains (e.g., useless check-in desks) makes it possible to manage this by imposing that a task must be equal to the following task minus one and by not including useless check-in desks in the domain. In this way, we insert a hole representing the zone’s separation. Constraints (4) prevent two overlapping registration from being assigned to the same check-in desk (as presented in Section 2.2). Constraints (5) use table constraint to map the check-in desk with this weight. We note \( w^c_\rho \) the weight associated to the check-in desk \( c \) considering the strategy rule of \( \rho \).

Finally, for constraints presented in Section 2.3 we use conflict tables.

### 3.2 Gathering Binary Difference Constraints

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

\[
\text{AllDifferentExcept}([x[\rho_1], x[\rho_2]], f) \tag{6}
\]

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

### 3.3 Gathering \textit{AllDifferentExcept} constraints

Even though the formulation above notably reduces the number of posted constraints, the solver remains too slow for finding acceptable results (bounds) in a reasonable amount of time. We have thus gathered all \textit{AllDifferentExcept} constraints into a unique pragmatic constraint called \textit{GatherAllDifferentExcept}. For this particular constraint, we use a specific fast propagator that performs a limited form of filtering (i.e., does not enforce generalized arc consistency). This is a very pragmatic approach, which is somewhat equivalent to the initial set of binary constraints, but quite faster (only one constraint being posted).
Table 1 Description of the main instances ($\Phi$) uses in this study.

| Instance          | $|C|$ | # Tasks | $|O|$ | # 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              |

4 Experiments Results

4.1 Instances

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

4.2 Decomposition

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

4.3 Results

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

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

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

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

We can see that the value-heuristic Static finds a good solution and reduces the number of registrations that are not assigned. However, the time needed to find this solution is higher than the one obtained with the Static formulation.

4. https://github.com/crillab/aceurancetourix
than BIVS, which finds an acceptable solution. The combination of both does not allow to reduce time or reduce the number of unassigned registrations. We can also observe that in the case of the Static configuration, GatherAllDifferent brings a 22% gain and reduces unallocated registrations by 45 in an acceptable amount of time.

5 Conclusion

In this paper, we have been interested in the Airport Check-in Desk Problem as defined at Paris Airport. We have proposed a COP model for this problem, mainly exploiting table constraints and developing an ad-hoc constraint (for reducing the number of constraints, and accelerating the resolution consequently). We have presented a first empirical evaluation of our approach. Our results look quite promising, as the ADP group starts replacing their current proprietary solution with ours, based on generic open-source tools (modeling library and constraint solver). In the future, we plan to simplify the execution process by limiting data processing in the PyCSP$^3$ model and discarding the decomposition step. We also plan to use a multi-criteria objective.

References