Modelling Robust Flight-Gate Scheduling as a Clique Partitioning Problem

Ulrich Dorndorf
INFORM GmbH, 52076 Aachen, Germany, ulrich.dorndorf@inform-ac.com

Florian Jaehn, Erwin Pesch
Institute of Information Systems, University of Siegen, Siegen, 57068, Germany
{florian.jaehn@uni-siegen.de, erwin.pesch@uni-siegen.de}

This paper considers the problem of assigning flights to airport gates. We examine the general case in which an aircraft serving a flight may be assigned to different gates for arrival and departure processing and for optional intermediate parking. Restrictions to this assignment include gate closures and shadow restrictions, i.e., the situation in which certain gate assignments may cause the blocking of neighboring gates. The objectives include maximization of the total assignment preference score, minimization of the number of unassigned flights during overload periods, minimization of the number of tows, as well as maximization of the robustness of the resulting schedule with respect to flight delays. We are presenting a simple transformation of the flight-gate scheduling (FGS) problem to a graph problem, i.e., the clique partitioning problem (CPP). The algorithm used to solve the CPP is a heuristic based on the ejection chain algorithm by Dorndorf and Pesch [Dorndorf, U., E. Pesch. 1994. Fast clustering algorithms. ORSA J. Comput. 6 141–153]. This leads to a very effective approach for solving the original problem.

Key words: airport gate scheduling; clique partitioning; robustness; ejection chain

History: Received: February 2006; revision received: December 2006; accepted: June 2007. Published online in Articles in Advance March 24, 2008.

1. Introduction

Gate scheduling is about finding an assignment of flights to terminal or ramp positions called gates. It is a key activity in airport operations. With the increase in civil air traffic and the corresponding growth of airports in the past decades, the complexity of the task has increased significantly. At large international airports, several hundred flights must be handled per day. The task is further complicated by frequent changes to the underlying flight schedule on the day of operations, such as delays or aircraft changes.

Gates are scarce and expensive resources. Increasing the resource supply by involving a time-consuming and costly redesign of terminal buildings or ramps is usually not feasible in the short term. It is therefore of great economic importance for an airport or terminal operator to use the available gates in the best possible way.

The main input for gate scheduling is a flight schedule with flight arrival and departure times and additional detailed flight information, including pairwise links between successive flights served by the same aircraft, the type of aircraft, the number of passengers, the cargo volume, and the origin or destination of a flight, classified, e.g., as domestic or international. The information in the flight schedule defines the time frame for processing a flight and the subset of gates to which it can or should be assigned, taking into account, e.g., aircraft–gate size compatibility, access to governmental inspection facilities for international flights, etc.

It is worth mentioning, from a practical point of view, one of the most important issues of gate scheduling: A gate schedule should be insensitive to small changes of input data; in other words, schedule flexibility is required. Obviously, the input data of any FGS problem are subject to uncertainty and may change over time.

Input data uncertainty in gate scheduling may arise from (1) flight or gate breakdown, (2) flight earliness or tardiness, (3) emergency flights, (4) severe weather conditions, (5) errors made by staff, or for many other reasons. For example, a tardy arrival of one aircraft may generate a chain of delayed arrivals for other aircraft that have been assigned to the same gate. In the worst case, this may lead to a “domino effect” and finally require a complete rescheduling, a situation that is highly undesirable.

In contrast to former approaches where gate scheduling has, for instance, been modelled either as a quadratic assignment problem or as a project scheduling problem, we will introduce an approach that, to our knowledge, has not been considered in the literature. Here, gate scheduling is modelled
as a CPP. The CPP is NP-complete, just like the quadratic assignment problem and the various project scheduling problems. However, there are not only efficient heuristics for solving the CPP; this approach also makes it possible to easily integrate a robustness goal, which makes the assignment more insensitive to flight earliness or tardiness. Robustness means, in this context, that there is a minimum buffer time between any two flights assigned to the same gate. All pairs of flights that do not achieve this minimum buffer time have a lack of buffer time. The sum of those lacks is the measure of the robustness goal.

Mathematical models of assigning flights to gates have been presented, e.g., by Ding et al. (2004) and Dorndorf et al. (2007). However, a robustness goal has not yet been included in these approaches. Considering the buffer time between two activities at a gate has been proposed by Hassounah and Steuart (1993) and Bolat (2000). A survey about robustness goals can be found in the study by Nikulin (2006).

The remainder of this paper is organized as follows. A formal definition of the problem and a related mathematical model are given in §2. An equivalent transformation of this model to the CPP is proposed in §3. Section 4 describes an ejection chain algorithm for solving the CPP. Computational experiments and comparisons are presented in §5. Finally, a summary appears in §6.

2. Problem Description

We are concerned with the task of assigning a set of flights, or activities, to the available gates. For this situation, we consider a relation

\[ f : N \rightarrow M \quad N := \{1, \ldots, n\}, \quad M := \{n + 1, \ldots, n + m\} \]

that maps the \( n \) activities onto the gate set \( M \). An activity \( i \in N \) can be an arrival, a parking, or a departure activity. Each activity may be assigned to a different gate, which would be handled in real life by towing the aircraft. If \( i, j, q \in N \) represent the arrival, parking, and departure of an aircraft, respectively, then we call \( j \) and \( q \) the successors of \( i \) and \( j \), respectively. If there is no parking activity \( j \), we call \( q \) the successor of \( i \). Because each activity has at most one successor we can define the function

\[ U : N \rightarrow N \cup \{0\} \]

\[ U(i) = \begin{cases} j & \text{if } j \text{ is successor of } i \\ 0 & \text{otherwise}. \end{cases} \]

The assignment \( f : N \rightarrow M \) is subject to some restrictions:

1. Gates may be closed for certain flights. Thus, the range of \( f \) within the codomain is restricted by

\[ f(i) \in M(i) \subseteq M \quad \forall i \in N. \]

These restrictions are motivated by aircraft that are too big for some gates, or by other technical factors such as gates that exclusively serve national or international flights. \( M(i) \) is not empty and includes at least \( n + m \) for every \( i \). Gate \( n + m \) can be regarded as a fictitious or dummy gate with unlimited capacity. Because the dummy gate does not correspond to a real aircraft position, dummy gate assignments will have to be eliminated for practical purposes.

2. For each pair of flight activities, there is information about a buffer time between activities or how far activities overlap. If two activities overlap, they are not allowed to be assigned to the same gate. However, two flight activities may always be assigned to dummy gate \( n + m \). The buffer and the overlap can be summarized in a symmetric \( n \times n \) matrix \( T \), where \( t_{ij} \geq 0 \) describes buffer time, whereas \( t_{ij} < 0 \) implies an overlap of \( |t_{ij}| \) time units. Thus, \( f \) must satisfy

\[ f(i) \neq f(j) \quad \forall t_{ij} < 0, \quad f(i) \neq n + m, \quad \forall i, j \in N. \quad (1) \]

In practical applications, the buffer time or the overlap time is taken from a timetable with a starting and finishing time for each activity. The buffer times in \( T \) can also reflect flight-dependent setup times, which may prohibit an assignment of two activities to the same gate. Furthermore all \( t_{ij} \) with \( U(i) = j \) are by definition set to a maximum value \( t_{\text{max}} \), which will be explained in more detail at the end of this section.

3. There is a set \( S \subset N \times M \times N \times M \) of so-called shadow restrictions, which prohibit the specific assignment of two flight activities to certain gates. Each element of \( S \) is a quadruple \( s = (i, k, j, l) \in S \), which restricts the assignment \( f(i) = k \) in combination with \( f(j) = l \)

\[ f(i) \neq k \vee f(j) \neq l \quad \forall (i, k, j, l) \in S. \quad (2) \]

A major reason for the shadow restrictions is that two big aircraft cannot be assigned to neighboring gates without wing tip overlap. Consequently, these restrictions do not apply to the dummy gate \( n + m \) and only apply to flight activities that overlap in time.

All of these restrictions can be trivially satisfied by assigning every flight activity to the dummy gate. However, this is an undesirable solution.

The objective function

\[ g(f) = \alpha_1 z_1(f) + \alpha_2 z_2(f) + \alpha_3 z_3(f) \]

to be minimized is a linear combination of the single objectives \( z_1, z_2, z_3 \) with given weights \( \alpha_1, \alpha_2, \alpha_3 \geq 0 \):  
1. For every assignment of a flight activity to a gate \( i \mapsto f(i) \in M \) there is an integer valued preference score. These preferences are obtained from an \( n \times m \)
matrix $P$, where $p_{ik}$ denotes the preference value of assigning flight activity $i$ to gate $k$. It follows that

$$z_1 := -\sum_{i=1}^{n} p_{ij(i)}.$$

These preferences include priority values, for example, for bigger aircraft, and they are determined by aircraft managers. Because dummy gate assignments are undesirable, we can assume that

$$p_{i(n+m)} < p_{ik} \quad \forall i \in N, k \in M(i) \setminus \{n + m\}.$$

If for a given $i \in N$, all $p_{ik}, k \in M$ are increased or decreased by a constant, $z_1$ is also changed only by a constant. We can therefore calculate $z_1$ with the help of

$$p_{ik}^* := p_{ik} - p_{i(n+m)} \quad \forall i \in N, k \in M(i)$$

because

$$z_1 := -\sum_{i=1}^{n} p_{ij(i)} = -\sum_{i=1}^{n} p_{i(n+m)} - \sum_{i=1}^{n} p_{ij(i)}.$$

Note that $p_{i(n+m)}^* = 0 \forall i \in N$.

2. As mentioned before, a flight activity is an arrival, parking, or departure of an airplane. If an activity and its successor are assigned to different gates, then a tow is necessary. So a tow is caused if and only if $U(i) = j > 0$ and $f(i) \neq f(j)$. It is desirable to keep the number of tows low. This leads to

$$z_2 := ||i \in N | U(i) \neq 0 \land f(i) \neq f(U(i))||.$$

3. Restrictions (1) guarantee that only nonoverlapping flight activities may be assigned to the same gate. Besides answering the question of whether two activities overlap, it is, furthermore, reasonable to keep a buffer time. A small buffer time between two activities that are assigned to the same gate may cause trouble if the first activity is delayed. Thus, a further objective is to avoid the assignment of two flights with low buffer times to the same gate. Low buffer time is defined by a parameter $t_{\max} \in \mathbb{N}$ so that buffer time $t_{ij} \geq 0$ is low, if and only if $t_{ij} < t_{\max}$. Recall that the buffer time of two successive flight activities is always set to $t_{\max}$. Thus, we get

$$z_3 := \sum_{\{i, j | i < j, f(i) = f(j) \neq n + m\}} \max\{t_{\max} - t_{ij}, 0\}.$$

Using a sum criterion within the third part of the objective measures the total buffer time. Its value might be reasonable although some of the assignments are not desired. Such an undesirable situation can be changed manually because our gate assignment algorithm or other rule-based systems are used only for decision support. The final decision is left to the airport planner.

The assignment problem can be summarized in the following optimization model

$$\min_{\{f, f: N \rightarrow M\}} \alpha_1 z_1(f) + \alpha_2 z_2(f) + \alpha_3 z_3(f)$$

s.t. $f(i) \in M(i) \quad \forall i \in N$

$$f(i) \neq f(j) \quad \forall i \in N, j \neq n + m$$

$$f(i) \neq k \lor f(j) \neq l \quad \forall (i, k, j, l) \in S$$

$$z_1 = -\sum_{i=1}^{n} p_{ij(i)}$$

$$z_2 = |\{i \in N | U(i) \neq 0 \land f(i) \neq f(U(i))\}|$$

$$z_3 = \sum_{\{i, j | i < j, f(i) = f(j) \neq n + m\}} \max\{t_{\max} - t_{ij}, 0\}.$$ (3)

Dorndorf (2002) analyzes a similar model that varies in two points: First, the robustness objective $z_2$ is not considered. Second, he uses further decision variables that allow varying the starting and finishing time of parking activities. This relaxes restrictions (1), because some overlaps can be avoided by shortening the parking activity and simultaneously lengthening either the arrival or the departure. Dorndorf obtains remarkable results solving his model using a layered branch-and-bound algorithm in combination with constraint propagation techniques.

The weights $\alpha_1$, $\alpha_2$, and $\alpha_3$ of the three objectives are determined by airport or airline managers. The principle idea is that for every airport based on representative data a set of Pareto optimal solutions has been generated. Based on the obtained Pareto frontier, the weights have been chosen in the most suitable way. The generation of all Pareto optimal solutions has not been considered as a feasible alternative because each solution requires a time-consuming manual inspection to judge its quality.

3. Clique Partitioning Problem

In this section we present an equivalent transformation of the optimization model (3) for the special case $S = \emptyset$ (no shadow restrictions) to the CPP.

Consider a complete, undirected, weighted graph $G = (V, E, W)$ with vertex set $V = \{1, 2, \ldots, a\}$, edge set $E \subseteq V \times V$, consisting of all two-element subsets $\{i, j\}, i \neq j$ of the vertex set $V$, and their symmetric edge weights $W = \{w_{ij}, i, j \in V, w_{ij} = w_{ji} \in \mathbb{R} \cup \{-\infty\}\}$. The CPP is to find an equivalence relation on $V$ so that the sum of the edge weights of all vertices in relation is maximized. This is equivalent to finding a partition of $V$ into cliques, i.e., a vertex subset, so that the sum of the edge weights within the cliques is maximized. Unless all weights are positive or all weights are negative, the CPP is NP-complete (Dyer...
and Frieze 1985; Grötschel and Wakabayashi 1990, 1989). With binary variables

$$x_{ij} = \begin{cases} 
1 & \text{if vertices } i \text{ and } j \text{ are in relation (i.e. } i \text{ and } j \text{ belong to the same clique),} \\
0 & \text{otherwise}
\end{cases}$$

for all edges $[i, j] \in E$, the CPP can be described by the following model (see Dorndorf and Pesch 1994)

$$\begin{align*}
\max & \sum_{1 \leq i < j \leq a} w_{ij} \cdot x_{ij} \\
\text{s.t.} & \quad x_j + x_k - x_{jk} \leq 1 \quad \text{for } 1 \leq i < j < k \leq a \\
& \quad x_j - x_k + x_{jk} \leq 1 \quad \text{for } 1 \leq i < j < k \leq a \\
& \quad -x_j + x_k + x_{jk} \leq 1 \quad \text{for } 1 \leq i < j < k \leq a \\
& \quad x_j \in \{0, 1\} \quad \text{for } 1 \leq i < j \leq a.
\end{align*}$$

(4)

The constraints guarantee the transitivity of the relation: If vertices $i$ and $j$ belong to the same clique and vertices as $j$ and $k$, then vertices $i$, $j$, and $k$ belong to the same clique.

Using the notation of the FGS problem, we will discuss an instance of the CPP. We consider $a := n + m - 1$ vertices. The first $n$ vertices are associated with the flight activities, whereas the next $m - 1$ vertices correspond to real gates, excluding the dummy gate. We define an equivalence relation on this set of vertices as a matching of flight activities to gates as follows: If a vertex $i \leq n$ is in relation to a vertex $k > n$, we say flight activity $i$ has to be assigned to gate $k$. If a vertex $i$ is not in relation to any vertex greater than $n$, we assign flight activity $i$ to the dummy gate. Choosing the edge weights appropriately will prohibit any two vertices from being related. This guarantees that every vertex $i \leq n$ is related to one vertex greater than $n$ at the most; i.e., every flight activity is assigned to exactly one gate, including the dummy gate. Figure 1 shows a situation with five flight activities and two gates. Vertices 6 and 7 symbolize gates, and bold edges define two vertices to be in relation. Flight activities 3 and 4 are assigned to gate 7, whereas flight activities 1, 2, and 5 are assigned to the dummy gate, because they did not obtain any particular gate assignment.

Modelling the gate assignment as a CPP instance, we will define the set of vertices by

$$V := \{1, 2, \ldots, n + m - 1\},$$

(5)

and the edge weights are defined as follows

$$w_{ij} := \begin{cases} 
-\infty & \text{if } t_{ij} < 0 \\
\alpha_2 & \text{if } t_{ij} \geq 0 \land (U(i) = j \lor U(j) = i) \\
-\alpha_3 \cdot \max\{t_{\max} - t_{ij}, 0\} & \text{if } t_{ij} \geq 0 \land U(i) \neq j \land U(j) \neq i \\
0 & \text{otherwise}
\end{cases} \quad \forall i, j \leq n$$

(6)

$$w_{ij} := \begin{cases} 
-\infty & \text{if } j \notin M(i) \\
\alpha_1 \cdot p_{ij} & \text{if } j \in M(i) \\
0 & \text{otherwise}
\end{cases} \quad \forall i \leq n, j > n$$

(7)

$$w_{ij} := -\infty \quad \forall i, j > n.$$ 

(8)

Equation (6) defines the edge weight of vertices representing any two arrival, departure, or parking activities. Overlapping activities, where $t_{ij} < 0$, may not be assigned to the same gate. Consequently, their edge weights are set to negative infinity. Nonoverlapping flight activities might be successors and operated by the same aircraft. If such activities are assigned to the same gate, a tow is saved. Their edge weight is set to $\alpha_2$. So, in this model we do not punish a necessary tow, but we reward avoided tows. The remaining nonoverlapping pairs of flight activities are either weighted with 0 if there is enough buffer time, or with $-\alpha_3 \cdot (t_{\max} - t_{ij})$ otherwise. The same transformation is possible if only the $k$ worst buffer time differences are valid. The worst situation is a situation that cannot be realized in practice, e.g., if the buffer time is not sufficient. The system should not propose such solutions because these conflicts must be resolved manually.

Equation (7) describes the weights of flight activities assigned to gates. If a flight activity must not be assigned to a particular gate, i.e., if $j \notin M(i)$, the weight is set to $-\infty$. Otherwise, the weight is set to the preference score $\alpha_1 \cdot p_{ij}$. There is no vertex representing the dummy gate, and so there are no edges to be weighted with $p_{ij}$. However, because $p_{ij} = 0$, $\forall i \in N$, objective $z_1$ is correctly adopted to the CPP, although flight activities are assigned to the dummy gate if and only if they are not in relation to any gate-representing vertex.

Finally, the weight of an edge connecting any two vertices representing gates is set to negative infinity (Equation (8)). This prohibits two gates from belonging to the same clique.
We refer to a solution of the CPP as feasible if and only if its objective is greater than \(-\infty\). In the following, we will build a bijection from the set of feasible solutions of the FGS formulation (3) to the set of feasible solutions of the CPP formulations (4) to (8). To build such a bijection it is necessary that both sets have same cardinality. However, there are more feasible solutions to the CPP than there are feasible solutions to formulation (3). The reason is that the graph model distinguishes whether two flight activities are assigned to the same dummy-gate clique or to different dummy-gate cliques. For example, the vertices 1 and 2 in Figure 1 belong to the same clique and both are assigned to the dummy gate. However, if these two vertices were not in the same clique, they could still be assigned to the dummy gate.

Observe that for any two flight activities representing vertices \(i\) and \(j\), their edge weight is only positive if they are succeeding; i.e., \(U(i) = j\) or \(U(j) = i\). Therefore, we require that succeeding flight activities must be assigned to the same dummy-gate clique and non-succeeding flight activities must not be assigned to the same dummy-gate clique. In this case, same cardinality required for a bijection is guaranteed.

**Theorem 1.** Consider the FGS problem (3) with \(S = \emptyset\) and the CPP described by (4) to (8). There is a bijection between the sets of feasible solutions of the two problems so that every feasible solution \(f\) of the FGS problem corresponds to one and only one feasible solution \(x\) of the CPP and vice versa. The objective value of \(f\) differs from the objective value of \(x\) only by the sign and by a constant.

**Proof.** The corresponding solution to a given \(f\) is defined by

\[
\begin{cases} 
1 & \text{if } f(i) = f(j) \neq n + m, \quad i, j \leq n \\
& \text{or } f(i) = f(j) = n + m \land (U(i) = j \lor U(j) = i), \quad i, j \leq n \\
& \text{or } f(i) = f(j) = n + m \land ((U(i) = h \land U(h) = j) \\
& \lor (U(j) = h \land U(h) = i), \quad i, j, h \leq n \\
0 & \text{otherwise.}
\end{cases}
\]

Given a feasible solution \(x\), for every \(i \in N\) there is at most one \(j > n\) so that \(x_{ij} = 1\). If there were \(j_1 > n\) and \(j_2 > n\) with \(x_{ij_1} = x_{ij_2} = 1\), then \(x_{ij_1} = 1\), which is a contradiction to the feasibility of \(x\). This allows the following definition of the corresponding solution to \(x\)

\[
f(i) := \begin{cases} 
j & \text{if } x_{ij} = 1, \quad j > n \\
n + m & \text{otherwise}
\end{cases}
\]

for all \(i \leq n\). Before showing the linear transformation of the objective, let us define the set of vertices assigned to the dummy gate, the set of vertices assigned to real gates, the set of succeeding pairs of vertices, and the set of not succeeding pairs of vertices assigned to the same gate

\[
N_1 := \{ i \in N \mid f(i) = n + m \} \\
= \{ i \in N \mid x_{ij} = 0, \forall j > n \}, \\
N_2 := N \setminus N_1, \\
U_1 := \{(i, j) \in N \times N \mid i < j, \ U(i) = j \lor U(j) = i \}, \\
U_2 := \{(i, j) \in N \times N \mid i < j, \ U(i) \neq i \} \\
U(j) \neq i, \ f(i) = f(j) \\
= \{(i, j) \in N \times N \mid i < j, \ U(i) \neq j \} \text{ and } U(j) \neq i, \ x_{ij} = 1 \}.
\]

We will now take advantage of the fact that \(x_{ij} = 0\) whenever \(w_{ij} = -\infty\). Transforming the objective of \(x\) leads to

\[
\sum_{1 \leq i < j \leq n + m-1} w_{ij} \cdot x_{ij} = \sum_{1 \leq i < j \leq n} w_{ij} \cdot x_{ij} + \sum_{1 \leq i < j \leq n \neq m-1} w_{ij} \cdot x_{ij} + \sum_{n < i < j \leq n - m - 1} w_{ij} \cdot x_{ij} \\
= \sum_{(i, j) \in U_1} \alpha_2 x_{ij} + \sum_{(i, j) \in U_2} -\alpha_3 \cdot \max\{t_{ij} - t_{ij}, 0\} \\
+ \sum_{i \in N_2} \alpha_1 (p_{f(i)} - p_{f(n + m)}) \\
= \alpha_2 |\{(i, j) \in U_1 \mid x_{ij} = 1\}| - \alpha_3 z_3 + \alpha_1 \sum_{i \in N \setminus N_2} p_{f(i)} \\
= \alpha_2 (|U_1| - |\{(i, j) \in U_1 \mid x_{ij} = 0\}|) - \alpha_3 z_3 + \alpha_1 \sum_{i \in N} p_{f(i)} \\
= \alpha_2 |U_1| - \alpha_2 z_2 - \alpha_3 z_3 - \alpha_3 z_1 \\
= \alpha_2 |U_1| - g(f).
\]

The objectives differ only between the sign and the constant \(\alpha_2 |U_1|\). □

Consequently, an optimal solution of the CPP corresponds to an optimal solution of the FGS. Therefore, it is sufficient to solve the CPP instead of model (3). The CPP is NP-complete, but there are efficient heuristics that can also integrate the shadow restrictions (2). Such a heuristic is presented in the following section.

**4. An Ejection Chain Algorithm**

We will present an algorithm to solve the CPP as described in (4) and (5) to (8). Furthermore, this algorithm satisfies any set of shadow restrictions as described by (2). We divide the algorithm into three subalgorithms in which Algorithm 1 is the core, Algorithm 2 defines the initial solution, and Algorithm 3
is the main algorithm, which restarts Algorithm 1 with different initial solutions. Algorithm 1 is basically the one presented in Dorndorf and Pesch (1994) with some modifications for the application at hand. Dorndorf and Pesch (1994), in turn, have based their algorithm on the ejection chain idea of Glover (1996) and Glover and Pesch (1997), and the variable-depth local search approach of Kernighan and Lin (1970).

From now on we regard a solution to the CPP as feasible only if its objective function value is greater than $-\infty$ and if, additionally, the corresponding solution of the FGS does not violate the shadow restrictions.

An equivalence relation on the set $V$ divides the vertices into clusters, the so-called equivalence classes or cliques. A single vertex defines a one-element cluster. The number of clusters is restricted by $n + m - 1$. We consecutively number these clusters, including those that are potentially empty. A basic principle of the algorithm is one-step moves. In each one-step move, a feasible solution $x$ is modified to a feasible neighbor solution $x'$. This is achieved by moving a vertex representing a flight activity $i$ from its current cluster $C \subset V$ to a new, possibly empty cluster $D \subset V$. Let us formalize the term “neighbor.”

**DEFINITION 1.** Let $x$ be a feasible solution of the CPP. We call a feasible solution $x'$ a neighbor of $x$ if there is an $i \leq n$ so that
1. $x'_{jk} = x_{jk}$ for every $j, k \neq i$
2. $x' \neq x$.

Condition 1 prohibits any changes to the relation unless vertex $i$ is involved, whereas condition 2 ensures that if $i$ is the only element of a cluster, then it may not be moved to an empty cluster.

The effect on the objective of moving $i$ from its current cluster $C(i)$ to a cluster $D$ can easily be calculated by

$$h(i, C(i), D) := \sum_{1 \leq i' < j \leq m} w_{ij} x'_{ij} - \sum_{1 \leq i' < j \leq m} w_{ij} x_{ij}$$

$$= \sum_{j \in D} w_{ij} - \sum_{j \in C \setminus \{i\}} w_{ij}.$$  

Such a move leads to a feasible neighbor, if and only if
1. $h(i, C(i), D) > -\infty$.
2. Moving $i$ to $D$ does not lead to a violation of the shadow restrictions.
3. $D = \emptyset \Rightarrow C(i) \neq \emptyset$.

The first two conditions are necessary and sufficient for the feasibility of $x'$, whereas condition 3 ensures that $x \neq x'$. Under these conditions, only feasible neighbors are reached, and therefore we always stay in the feasible region.

Starting from a feasible solution, our algorithm greedily searches for a sequence of at most $n$ moves. Each move maximizes $h(i, C(i), D)$ under the restriction that a vertex must not be moved more than once in this sequence. Practically, this means that every flight activity is moved from its current gate to another one. However, to reach a feasible solution for the next iteration, only the subsequence is performed, which raises the objective at most. If there is no such subsequence, the algorithm will stop.

**ALGORITHM 1.**

0. **Initialization:** Start with a feasible solution.
1. **Finding a sequence of $n$ moves:** Duplicate the current solution and denote all $n$ flight activity representing vertices as nontabu.

   1.1 **Finding a best move:** Find a move among all nontabu vertices, which leads to a feasible neighbor and which maximizes $h(i, C(i), D)$. Perform this move to the duplicated solution, mark the moved vertex $i^*$ as tabu, and update all clusters.

   Repeat 1.1 to the duplicated solution until all vertices are marked tabu or until no feasible neighbor can be found.

   Determine $r \in [0, \ldots, n]$ so that the duplicate’s objective is maximized after performing the first $r$ moves and improves the objective of the current solution.

2. **Improving the solution:** If $r > 0$, perform the first $r$ moves to the current solution and go to step 1. If $r = 0$ then return the current solution.

The advantage of at least intermediatively performing $n$ moves is that we can escape local minima. Therefore, the maximal $h(i, C(i), D)$ in step 1.1 may be negative.

To find the best move in step 1.1, at most $n \cdot (n + m - 1)$ moves have to be checked. Hence, the best neighbor is explored in time complexity $O(n^2)$ assuming that $n \geq m$. Because a sequence of $n$ moves is searched, the time complexity of a whole iteration is $O(n^3)$. However, as shown by Dorndorf and Pesch (1994), the time complexity of one iteration can be reduced to $O(n^2 \cdot \log n)$.

Algorithm 1 demands a (reasonable) initial solution. Although it is possible to consider a solution in which no two vertices are in relation, this bears a disadvantage for our special problem. With such an initial solution, the very first run through step 1 of the algorithm will deliver a greedy assignment of all flight activities to gates. Because this assignment is greedy, and because parking activities usually have lower preferences to gates than arrival or departure activities, the parking activities are assigned to gates last. At the point where a parking activity should be assigned to a gate, it is often not possible to choose the same gate as the corresponding arrival or departure activity because of other activities blocking the gate. Therefore, the solution obtained after the first iteration delivers a very high number of tows, which cannot be sufficiently reduced during further iterations.
Algorithm 2: Initial Solution.
0. Start with the feasible solution in which no two vertices are in relation.
1. Denote all flight activity representing vertices $i$ with $U(i) \neq 0 \lor U(U(i)) \neq 0$ (i.e., all arrivals) as nontabu and all others as tabu.
2. Among all nontabu vertices determine those $i^*$ and $D^*$ such that

\[
h(i, C(i), D) + h(U(i), C(U(i)), D) + h(U(U(i)), C(U(U(i))), D) > 0,
\]

is maximized. Denote $i^*$ as tabu.
3. If

\[
h(i^*, C(i^*), D^*) + h(U(i^*), C(U(i^*)), D^*)
\]

assign $i^*$, $U(i^*)$, and $U(U(i^*))$ to cluster $D^*$. If there are still nontabu vertices, then go to step 2.

The initial solution that we propose greedily assigns all triples of succeeding flights to a gate. If at some point a triple of succeeding flights cannot be assigned to any gate (because any assignment would lead to an unfeasible solution), all three flight activities are assigned to the dummy gate. Activities that have no successor and that are not the successor of any other activity (e.g., an arriving aircraft that stays at the airport) are assigned to the dummy gate, too.

On our real-life problems as they are described in the next section, Algorithm 1 terminates in less than half a minute. Thus, it might be reasonable to run the algorithm several times with various initial solutions to achieve even better results within a couple of minutes. With the additional runs we especially want to reduce the number of dummy gate assignments. These assignments have a special impact on the objective so it is particularly important to keep their number low. Some flight activities are mandatory assigned to the dummy gate. For example, this is the case if every gate is closed for some flight, or if there is a mandatory tow, but the tow time is too high.

If a solution generated by Algorithm 1 consists only of mandatory dummy gate assignments, additional runs are not performed. Otherwise, the initial solution of the next run is the slightly modified resulting solution of the previous run. The modification of the previous solution can be described as follows: Each nonmandatory dummy gate assignment $i$ will be moved to the clique of its most preferred gate. This will lead to an infeasible solution because Algorithm 1 would have performed this move. To obtain a feasible solution, all flight activities that are now in conflict with $i$, for example, $x_{ij} = 1$ but $w_{ij} = -\infty$, will then be assigned to the dummy gate.

To avoid endless additional runs they are stopped if an initial solution recurs or if the number of runs exceeds 7. The restarts are put together in Algorithm 3, which resorts to Algorithms 1 and 2.

Algorithm 3.
0. Initialization: Run Algorithm 2 to find an initial solution and use it to run Algorithm 1. Save the result as the current solution and as the best solution.
1. Stop criterion: If the current solution is better than the best solution, then save the current solution as the best solution. If the current solution only has mandatory dummy gate assignments or if this step is reached seven times, then stop and return the best solution.
2. Changing current solution: For all nonmandatory dummy gate assignments $i \in N$ do

2.1. Assigning $i$ to a gate: Set

\[x_{ij} = 0 \text{ if } U(i) = j \text{ or } U(j) = i\]

\[x_{jk} = 1 \text{ for a } k \in M: w_{jk} = \max_{k \in M} p_{ik}^*\]

2.1.3. Eliminate gate conflicts: $\forall j \in N: x_{ij} = 1 \land \exists i \in N: x_{ij} = 1$

2.2. Eliminate gate conflicts: $\forall j \in N: \exists(i,k,j,l) \in S: x_{ij} = 1 \land w_{jl} = -\infty$

2.3. Eliminate shadow conflicts: $\forall j \in N: w_{ij} = 0$

3. Restart: Use the current solution as initial solution of Algorithm 1. Save the result of Algorithm 1 as current solution and go to step 1.

5. Computational Results
Algorithm 3 has been implemented in Java 2 and tested using two sets ($A$ and $B$) of real-life data. The first test set $A$ consists of 14 records (1.08–1.14 and 2.03–2.09) where each record corresponds to a day of a two-week period. A major American airport with $m = 94$ gates, including a dummy gate, has been considered. The number of flight activities per day varies between 635 and 820. Each record consists of a set $M(i)$, a matrix $T$, a successor function $U$, a set of shadow restrictions $S$, and a preference value matrix $P$.

The 14 test records under consideration are shown in Table 1. The number of flight activities is split into single activities and triple activities. Each triple activity consists of an arrival, a parking, and a departure activity. Furthermore, the total processing time of all flight activities and the maximum number of flight activities being at the airport at the same time is shown in the last two columns. Table 2 shows the spread of processing lengths of the different kinds of
activities. It is easy to see that processing times of parking activities vary a lot and usually last longer than other activities.

The second test set consists of 10 records (I, II, . . ., X) in which each record represents a planning period of a different airport, except records VI and VII, which correspond to the same airport. We considered eight major European airports and one major Asian airport. Again, each record consists of a set \( M(i) \), a matrix \( T \), a successor function \( U \), a set of shadow restrictions \( S \), and a preference value matrix \( P \). The details of test set B are described in Tables 3 and 4. Test set B contains records from different airports; therefore, the third column of Table 3 shows the number \( m \) of available gates. In this data set, it might happen that there are arrival and departure activities without a corresponding parking activity. In some records, the length of few activities was intentionally set to one minute.

### Table 1: The 14 Records of Test Set A

<table>
<thead>
<tr>
<th>Data</th>
<th>( n )</th>
<th>Single activities</th>
<th>Triple activities</th>
<th>Processing (hours)</th>
<th>Simultaneous flights (peak)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1_08</td>
<td>735</td>
<td>114</td>
<td>207</td>
<td>730</td>
<td>45</td>
</tr>
<tr>
<td>1_09</td>
<td>801</td>
<td>90</td>
<td>237</td>
<td>756</td>
<td>45</td>
</tr>
<tr>
<td>1_10</td>
<td>783</td>
<td>93</td>
<td>230</td>
<td>726</td>
<td>45</td>
</tr>
<tr>
<td>1_11</td>
<td>806</td>
<td>86</td>
<td>240</td>
<td>765</td>
<td>46</td>
</tr>
<tr>
<td>1_12</td>
<td>797</td>
<td>89</td>
<td>236</td>
<td>728</td>
<td>45</td>
</tr>
<tr>
<td>1_13</td>
<td>800</td>
<td>92</td>
<td>236</td>
<td>761</td>
<td>44</td>
</tr>
<tr>
<td>1_14</td>
<td>666</td>
<td>54</td>
<td>204</td>
<td>987</td>
<td>57</td>
</tr>
<tr>
<td>2_03</td>
<td>723</td>
<td>132</td>
<td>197</td>
<td>1,059</td>
<td>51</td>
</tr>
<tr>
<td>2_04</td>
<td>820</td>
<td>85</td>
<td>245</td>
<td>963</td>
<td>52</td>
</tr>
<tr>
<td>2_05</td>
<td>799</td>
<td>82</td>
<td>239</td>
<td>990</td>
<td>53</td>
</tr>
<tr>
<td>2_06</td>
<td>818</td>
<td>80</td>
<td>246</td>
<td>1,015</td>
<td>55</td>
</tr>
<tr>
<td>2_07</td>
<td>815</td>
<td>80</td>
<td>245</td>
<td>977</td>
<td>54</td>
</tr>
<tr>
<td>2_08</td>
<td>818</td>
<td>77</td>
<td>247</td>
<td>1,103</td>
<td>54</td>
</tr>
<tr>
<td>2_09</td>
<td>635</td>
<td>59</td>
<td>192</td>
<td>1,056</td>
<td>57</td>
</tr>
</tbody>
</table>

### Table 2: Processing Times in Minutes of the Different Kinds of Activities (Test Set A)

<table>
<thead>
<tr>
<th>Type of activity</th>
<th>Minimum length</th>
<th>Maximum length</th>
<th>Average length</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arrival</td>
<td>10</td>
<td>75</td>
<td>18</td>
</tr>
<tr>
<td>Departure</td>
<td>15</td>
<td>75</td>
<td>25</td>
</tr>
<tr>
<td>Parking</td>
<td>1</td>
<td>7,373</td>
<td>177</td>
</tr>
<tr>
<td>Single</td>
<td>9</td>
<td>150</td>
<td>45</td>
</tr>
</tbody>
</table>

### Table 3: The 10 Records of Test Set B

<table>
<thead>
<tr>
<th>Data</th>
<th>( n )</th>
<th>( m )</th>
<th>Single activities</th>
<th>Double activities</th>
<th>Triple activities</th>
<th>Processing (hours)</th>
<th>Simultaneous flights (peak)</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>1,437</td>
<td>302</td>
<td>780</td>
<td>0</td>
<td>219</td>
<td>4,432</td>
<td>93</td>
</tr>
<tr>
<td>II</td>
<td>298</td>
<td>21</td>
<td>298</td>
<td>0</td>
<td>0</td>
<td>258</td>
<td>5</td>
</tr>
<tr>
<td>III</td>
<td>694</td>
<td>42</td>
<td>694</td>
<td>0</td>
<td>1,877</td>
<td>22</td>
<td></td>
</tr>
<tr>
<td>IV</td>
<td>899</td>
<td>307</td>
<td>686</td>
<td>0</td>
<td>71</td>
<td>6,263</td>
<td>64</td>
</tr>
<tr>
<td>V</td>
<td>394</td>
<td>124</td>
<td>121</td>
<td>0</td>
<td>91</td>
<td>698</td>
<td>51</td>
</tr>
<tr>
<td>VI</td>
<td>1,966</td>
<td>226</td>
<td>94</td>
<td>0</td>
<td>624</td>
<td>2,218</td>
<td>143</td>
</tr>
<tr>
<td>VII</td>
<td>1,119</td>
<td>226</td>
<td>448</td>
<td>139</td>
<td>131</td>
<td>2,218</td>
<td>129</td>
</tr>
<tr>
<td>VIII</td>
<td>553</td>
<td>263</td>
<td>293</td>
<td>13</td>
<td>78</td>
<td>1,370</td>
<td>89</td>
</tr>
<tr>
<td>IX</td>
<td>807</td>
<td>62</td>
<td>660</td>
<td>0</td>
<td>49</td>
<td>922</td>
<td>35</td>
</tr>
<tr>
<td>X</td>
<td>213</td>
<td>184</td>
<td>8</td>
<td>0</td>
<td>44</td>
<td>741</td>
<td>44</td>
</tr>
</tbody>
</table>
by airport managers. Thus, it is not reasonable to include minimum processing times in Table 4.

In our first test phase we used the solutions by the layered branch-and-bound algorithm described by Dorndorf (2002) for comparison. Note that it is a truncated branch-and-bound algorithm that does not explore the complete search tree. We chose the same parameters, and we split the preference value weight \(\alpha_1\) into \(\alpha_1(l)\) and \(\alpha_1(II)\). \(\alpha_1(l)\) denotes the preference value weights for dummy gate assignments, whereas \(\alpha_1(II)\) describes the preference value weight for real gate assignments. For test set A they are \(\alpha_1(l) = 400\), \(\alpha_1(II) = \alpha_2 = 1, \alpha_3 = 0\).

Table 5 displays the results of the 14 instances of test set A. Column \(z_1(l)\) shows the number of flight activities assigned to the dummy gate. The second entry of this column contains the number of mandatory assignments to the dummy gate. \(z_1(II)\) presents the preference scores, excluding dummy gate assignments. \(z_2\) refers to the number of tows needed in the solution, whereas \(g(f)\) represents the whole objective score.

As we can see, the ejection chain algorithm gets along with less dummy gate assignments, with similar number of tows and similar preference score. On average, its objective values are lower than the ones in the layered branch-and-bound algorithm. The computation times of the ejection chain algorithm\(^1\) and the layered branch-and-bound algorithm\(^2\) were similar and every instance was solved in less than five minutes.

It is furthermore worth mentioning that the layered branch-and-bound algorithm was only recently introduced at the airport. The rule-based gate scheduling approach that was used before and also tested on these 14 test sets assigned over 200 flight activities to the dummy gate, it needed over 1,100 tows, and the preference score amounted to \(-2,576\).

The parameters for test set B vary within the records. Table 6 shows these parameters, which are normalized to the preference weight \(\alpha_1(II)\).

Following Table 6, the results of test set B are listed in Table 7. Because the records describe different airports, the scores are not summed up in the last line. Again, the ejection chain algorithm has fewer dummy gate assignments. Newly, the number of tows is lower, but the preference score is worse in some instances. In the records in which the preference score is particularly relevant (II, IV, VIII, and IX), this leads to a similar or worse objective \(g(f)\). In all other records, the objectives are significantly improved again.

Apart from the objective score, the main advantage of modelling problem (3) as a CPP is the possibility to easily integrate the robustness goal. In our second test phase we integrated the robustness objective \(z_3\). We chose \(t_{max} := 30\) minutes and \(\alpha_3 := 0.1\). That means if two activities with zero buffer time are assigned to

---

1 Tests were run on a Pentium M, 1.99 GHz with Windows operating system.

2 Tests were run on a Pentium 3, 1.266 GHz with Linux operating system.
the same gate, the objective is raised by three (as it is, e.g., for three tows in test set A). In Tables 8 and 9 we show the results of our ejection chain algorithm after including the robustness objective. We also show the objective function value that is obtained if $\alpha_3 = 0$. It is remarkable that in test set A, although the robustness measure is considered, the overall objective value with $\alpha_3 = 0$ is still lower than the one of the layered branch-and-bound algorithm. The integration of the robustness goal also has little effect on the nonrobust objectives for test set B.

### 6. Summary

We have presented a transformation of the well-known FGS problem to the CPP. This leads to three major advantages: First, effective algorithms for the CPP are available that yield better results than the previous approaches to solving the FGS, as we have shown in our computational results. Second, this model converges through its simplicity. All information concerning the objectives and almost all restrictions are contained in a complete graph. As a consequence, the computation time is very short. Third, in contrast to most previous approaches, a robustness objective can be easily integrated. Such a robustness objective has only recently been considered in the literature, although its practical relevance is obvious. Computational results for real-world instances demonstrate the potential and superiority of the approach to frequently used rule-based systems currently in practice.

### Acknowledgments

The authors thank three anonymous referees for valuable comments and suggestions that improved the presentation of this paper. This work was supported by the German Science Foundation (DFG) through the Grant “Planung der Bodenabfertigung an Flughäfen” (PE 514/10-1).

### References


