CHAPTER

FOUR

Solving the gate assignment problem

Introduction

Between the time an aircraft lands at an airport and the time it departs again many things must happen. One of the most obvious things is that the passengers need to disembark the aircraft. Moreover, the aircraft needs to be refueled, new passengers need to board it, new supplies have to be put on board, and the aircraft has to get cleaned. These latter two actions are taken care of by the so-called *ground handler*. All of these actions take place while the aircraft is standing at a gate. We will refer to the arrival of an aircraft till the following departure of the same aircraft as a *flight*.

Now the *gate assignment problem* is described as follows: assign a given set of flights to a (possibly smaller) set of gates while making sure that certain criteria are met. Examples of such criteria are:

- Gates can handle only flights operated by aircraft of certain sizes.

- Gates can handle only flights for certain origins/destinations (e.g. because of safety regulations).
- Gates can handle only flights assigned to certain ground handlers.
- Two adjacent gates can not be assigned flights operated by big aircraft at the same time.
- Two adjacent gates should not be assigned flights with equal departure times.

Depending on the airport and its characteristics, many variants of the gate assignment problem have been researched and many solution methods have been suggested. A good overview of explored methods and models is given in Van Orden (2002). In this chapter we consider the situation at Amsterdam Airport Schiphol (AAS). Depending on the time horizon of the planning we distinguish between the following three planning problems: *seasonal planning*, *daily planning*, and *tactical planning*. The seasonal planning problem is a capacity planning problem. In this problem the gate planners must decide to accept or decline new requests from airlines to have their aircraft visit AAS. The daily planning concerns the creation of a planning for the upcoming day on the basis of the available information about the flights of that day. Finally, the tactical planning is an online planning problem. This problem concerns the resolving of conflicts that arise in the planned solution (i.e. the planning created the day before) due to disturbances.

The problem we consider in this chapter is daily planning. When looking at this planning problem, different objectives can be taken into account. Examples are minimizing total waiting time for the aircraft (i.e. the time an aircraft has to be held after landing before the gate is free) or minimizing total towing actions. Another objective that is used very often in the literature is to minimize the walking distance for the passengers to and from the gate (Bihr, 1990; Haghani and Chen, 1998; Xu and Bailey, 2001). Moreover, it is also possible to consider multi-criteria objectives. Dorndorf et al. (2007) not only present a state-of-the-art of the gate assignment problem in general but also discuss recent developments with regard to the multi-criteria objectives.

Since an airport is a very dynamic environment, an actual day will hardly ever go completely as planned; flights arrive either earlier or later than planned due to all kinds of reasons. The objective we are concerned with is to create a *robust*

schedule, i.e., a schedule that is able to cope with small perturbations in arrival or departure time without the need to replan big parts of the schedule. During meetings we had with the gate planners at AAS, finding a robust schedule was identified as the main target of their daily planning activities. Moreover, every change needed in the gate assignment during the actual day has an effect on a very broad range of parties: the ground handler, the security personnel, the passengers, etc.

Therefore, creating a schedule that needs fewer changes during the tactical planning is of great importance for a variety of parties. The same objective has been considered by Bolat (2000) and Van Orden (2002).

The question is how to measure the robustness of a schedule. The probability that a conflict arises between two consecutively scheduled flights at the same gate depends on the 'reliability' that the aircraft will stick to their assumed departure and arrival times and on the idle time between these two consecutive flights. The first part is outside the scope of the gate planners, and consequently we focus on optimizing the choice of the pairs of flights that we put consecutively at a gate. One approach for optimizing the individual idle times, and thus finding a robust schedule, is to minimize the variance of the idle time between successive flights at a gate. This approach is used in Bolat (2000) where the problem is formulated as an Integer Linear Program (ILP) using binary decision variables that link flights to positions at a gate. Based on this research a similar approach was followed in Van Orden (2002) for modelling the gate assignment problem for AAS. Although the suggested model showed promising results, a weak point of this model was that it was not possible to solve problems with more than 80 aircraft and 20 gates in a reasonable amount of time.

In this chapter we aim at finding a robust gate assignment schedule for a full day of traffic at AAS, which has about 600 flights. We include all real constraints occurring at the airport that we have identified in discussions with the gate-planners of the airport. To attain robustness, we optimize the idle time between all consecutive flights at the gates. Instead of minimizing the variance, we develop an objective function based on the arctangent, and we make some adjustments to this function to take the 'reliability' of the flights into account. We formulate this problem as an ILP and use a completely different representation from the one used in Van Orden (2002). This different representation is derived from the one used in Freling et al. (2001) for the vehicle and crew scheduling. We present an algorithm based on column generation to find a very

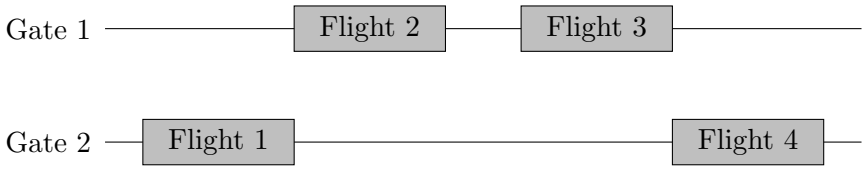

Figure 4.1: Example of a non-robust schedule.

good approximation for the optimum in this model. Our experiments indicate that this algorithm is able to solve real-life instances to near-optimality within a few minutes. Because of the high quality of the approximation even a near-optimal solution will suffice and techniques as branch-and-price (Barnhart et al., 1998) to solve the problem to full optimality will not pay off.

The outline for the remainder of this chapter is as follows: In Section 4.1 we give a detailed description of the gate assignment problem. In Section 4.2 we formulate the gate assignment problem as an ILP and present our algorithm to find an approximate solution. In Section 4.3 we will present the experimental results, and finally in Section 4.4 we draw some conclusions and indicate some future research topics.

4.1 Problem description

As mentioned in the previous section, our objective is to maximize the robustness of a solution to the gate assignment problem. To maximize the robustness we maximize the idle time between all pairs of consecutive flights at a gate, ensuring that each flight can arrive either a bit too early or a bit too late without the need for replanning the schedule. An example of a non-robust schedule can be found in Figure 4.1. This schedule does not have a lot of margin between flight 2 and flight 3. By modifying the schedule such that flight 3 is assigned to gate 2, while flight 4 gets assigned to gate 1 we introduce a lot more robustness in the solution.

One of the things that must be taken into consideration during the gate assignment process at AAS is whether two consecutive flights assigned to the same gate, are operated by the same airline or share the same ground handler. Such a situation is *convenient*, since the airline and the ground handler respectively will have an incentive to make the first flight leave on time. Furthermore, some airlines are known to be unreliable, meaning that if a flight of such an airline is due to depart at a certain time, then there is a great chance that it is delayed.

There are several hard constraints in the gate assignment problem. Obviously, each flight must be assigned to a gate, and two flights cannot be assigned to the same gate at the same time. But there are many more hard constraints, concerning the properties connected with the flights and the gates. The properties that are known for each flight are:

- The region of the origin of the flight,
- The region of the destination of the flight,
- The size category of the flight,
- The ground handler for the flight.

The region can be either Schengen (which refers to the countries that signed the Schengen Agreement), European Union (EU), or Non-EU. Often the region of the origin and the region of the destination of a flight are the same, but there are many exceptions, including e.g. transit flights that do not have AAS as their final destination. With respect to the size category of the flight, there are 8 categories at AAS: category 1 for the smallest aircraft up to category 8 for the biggest aircraft. Finally, the ground handlers are divided into two groups at AAS: KLM Ground Services, and all other companies.

For each of the gates it is known which regions, which size categories, and which ground handlers it can serve. When assigning flights to a certain gate, we need to satisfy the following three essential properties:

- The regions of origin and destination of the flight must match the possible regions of the gate.
- The size category of the flight must match the possible size categories of the gate.

- The ground handler of the flight must match the possible ground handlers of the gate.

One important issue of the gate assignment problem is that some of the flights stay at AAS for a longer period of time; for example, they arrive early in the morning and leave again late in the afternoon. If there are many such *long-stay* flights that stay at the gate, then the number of available gates quickly decreases. To circumvent this problem, the gate planners at AAS have the possibility of *splitting* the stay of long-stay flights into three different parts:

- *Arrival part.* After the aircraft lands, it will stay at the gate for 65 minutes, after which it is towed to some buffer stand.

- *Intermediate part.* During this part the aircraft resides on a buffer stand, where it does not use precious gate capacity.

- *Departure part.* The aircraft is taken from the buffer to the appropriate gate, 95 minutes before the aircraft will depart.

At AAS only flights that stay longer than three hours are considered for such a splitting. The advantages of splitting long-stay flights are three-fold. First, there is the obvious extra capacity that becomes available for the assignment of other aircraft. The second advantage concerns flights with different regions for the origin and destination: such flights normally would have to be assigned to a gate that is multi-purpose with respect to the region property, and these gates are quite scarce. When such a flight is split into parts, the separate parts do not have to be assigned to the same gate and thus can be assigned to two single region gates. Third, the decoupling of the parts yields additional flexibility.

Currently the process of splitting the long-stay flights is done manually. First the gate planners try to solve the gate assignment problem without splitting any flights. If the available capacity is not sufficient to accommodate all flights, the gate planners determine which flights should be split and then solve the problem again. This step is repeated several times until sufficient capacity is available.

4.2 Problem formulation

There are several options to value the idle time between two consecutive flights. Bolat (2000) considers minimizing the variance of the idle time. Here the idle times before the first flight and after the last flight on a gate are taken into account as well, which implies that the total idle time is a constant. As a result, minimizing the variance becomes equivalent to minimizing the sum of the squared idle times.

Another option is to define a cost function to value the idle time, which is the approach we use. This cost function for the idle time must reflect the natural appreciation of a solution. First of all, it should penalize very small idle-times with very high cost and it should only mildly penalize rather large idle times. Second, the function should be steep in the beginning (for small idle times) and then flatten out, to reflect that for small idle times any improvement is very beneficial, whereas for already large idle times an extra increase is of minor importance. Third, it should be possible to combine this with a refinement reflecting the *reliability* of a flight, which is discussed below. We found that a cost-function based on the arctangent fulfills the desired properties best. This function is defined as follows:

$$\mathrm{c}(t) = 1000(\arctan(0.21(5-t)) + \frac{\pi}{2}),$$

where t is the amount of idle time. Initial experiments showed that it was not beneficial to make use of a *cut-off value* (i.e. a threshold after which any increase in idle time will not result in a decrease of the costs). Such a cut-off value resulted in longer running times for the ILP, which can be explained by the fact that the cut-off value introduces a lot of symmetry in the problem.

Recall from the previous section there could be an advantage (or disadvantage) in assigning certain pairs of flights consecutively to the same gate. To model such a relation we introduce a convenience multiplier $conv(v, w)$ for flights v and w. To compute the cost of placing flight w immediately after flight v at the same gate, the cost corresponding to the idle time in between these flights will be multiplied by this multiplier. If putting flight w after flight v is desirable (e.g. flight v and w are operated by the same airline or handled by the same ground handler), then the convenience multiplier is given a value less than 1, thus decreasing the cost. On the other hand, inconvenient situations (e.g. when

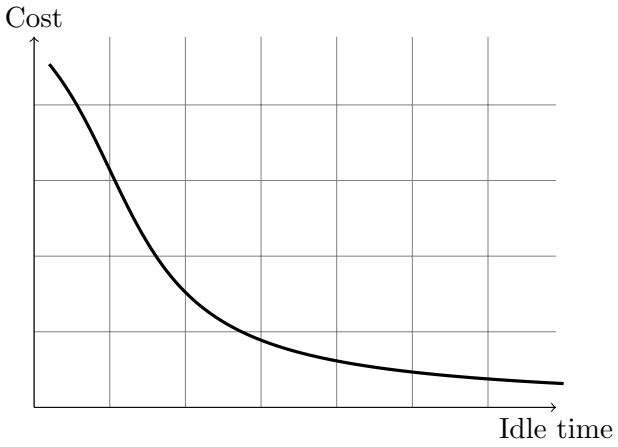

Figure 4.2: Cost function to value idle time.

flight v is operated by an unreliable airline) can be penalized by giving the multiplier for these situations a value greater than 1, thus increasing the cost of such an assignment. An advantage of this approach is that the convenience multiplier is computed beforehand for all possible pairs of successive flights, while during run-time only one additional multiplication per pair of consecutive flights is needed.

Since instances of the complete gate assignment problem consist of around 600 flights and 120 gates, we would like to see if we can decompose the problem into a set of smaller subproblems based on one of the properties associated with each gate (i.e. the possible regions, sizes, and ground handlers). Unfortunately for each of those properties gates exist that are multi-purpose for that particular property. This means that a strict division into multiple independent subproblems based on any of the properties is not possible without discarding available capacity: a given multi-purpose gate can only be assigned to one of the subproblems. Hence, such a static division is not desired.

The standard approach for modeling the problem as an ILP uses variables that denote whether a flight is assigned to a certain gate, see Van Orden (2002). One major disadvantage of this kind of model is that many additional variables are needed to determine the order of the flights, which is necessary to compute the idle time between two consecutive flights and hence the cost of a solution. More-

over, the number of constraints grows rapidly when relations between multiple gates and flights are present in the problem. This rapid growth of both variables and constraints makes it impossible to solve a gate assignment problem of the size that occurs at AAS. Therefore, a new approach is needed.

Based on the approach used by Freling et al. (2001) for the single-depot vehicle scheduling problem, we split the gate assignment problem into two phases. In the first phase we aggregate gates with the same properties (*identical gates*) into groups of gates and each such group we refer to as a *gate type*. We now introduce a *gate plan* as a series of flights that are to be assigned to the same gate of a certain gate type. Gate plans enable us to check feasibility easily: all flights present in the gate plan must satisfy the properties of the gate type that the gate plan corresponds to and no two flights should be assigned at the same time. Furthermore, calculating the cost of a gate plan boils down to summing the cost of all idle times between consecutive flights within the gate plan. With this representation solving the first phase comes down to finding a set of gate plans such that we have a gate plan for each physical gate, and that each flight is present in exactly one of the gate plans.

After we have obtained a solution for the first phase, for each gate type we have as many gate plans as there are gates of that gate type. In the second phase we undo the aggregation of identical gates into groups of gate types and we have to assign each gate plan to a physical gate.

4.2.1 Assigning flights to gate plans

When we look at the number of possible gate plans, we can clearly see that the number of possible gate plans is enormous. Suppose for the moment we do have the complete set of all possible gate plans, the gate assignment problem can be formulated as determining which of these gate plans we must select. Now for each gate plan i we introduce a decision variable x_i as follows:

$$x_i = \begin{cases} 1 & \text{if gate plan } i \text{ is selected} \\ 0 & \text{otherwise.} \end{cases}$$

Let V denote the number of flights, let A denote the number of gate types, let S_a denote the number of gates of type a, and let N denote the number of gate plans. Now the basic model for the gate assignment problem is as follows:

$$\text{Minimize} \sum_{i=1}^{N} c_i x_i$$

subject to:

$$\sum_{i=1}^{N} g_{vi} x_i = 1 \text{ for } v = 1, \ldots, V \tag{4.1}$$

$$\sum_{i=1}^{N} e_{ia} x_i = S_a \text{ for } a = 1, \ldots, A \tag{4.2}$$

$$x_i \in \{0, 1\} \text{ for } i = 1, \ldots, N \tag{4.3}$$

where

$$g_{vi} = \begin{cases} 1 & \text{if flight } v \text{ is in gate plan } i; \\ 0 & \text{otherwise,} \end{cases}$$

$$e_{ia} = \begin{cases} 1 & \text{if gate plan } i \text{ is of type } a; \\ 0 & \text{otherwise.} \end{cases}$$

We will extend this basic model to cover more possibilities. One important issue not addressed by the above model is the fact that it should be possible to give *preference* to placing a flight at a certain gate type. Such preferences can be used to model that certain flights are preferably assigned to certain gate types, e.g. because of the size of the waiting area or because of airlines having their "own" gates where at least a certain percentage of their flights have to be assigned to. Each preference consists of a set of flights, a set of gate types, and the minimum number of flights out of the given set that should be assigned to the given set of gate types.

Preferences can be modelled in the ILP by adding the following constraints to the model:

$$\sum_{i=1}^{N} \sum_{v=1}^{V} \sum_{a=1}^{A} p_{vak} e_{ia} g_{vi} x_i \geq P_k \text{ for } k = 1, \ldots, K \tag{4.4}$$

where

$$p_{vak} = \begin{cases} 1 & \text{if flight } v \text{ has preference for gate type } a \text{ in preference } k; \\ 0 & \text{otherwise,} \end{cases}$$

P_k denotes the minimum number of flights that has to be assigned to a given gate type according to preference k, e.g. an airline must have at least 6 flights at their "own" gates. K denotes the total number of preferences.

We extend the above model to deal with the case that there is not enough capacity to accommodate all flights. To solve this problem we add a penalty variable UAF_v (unassigned flight) for every flight v to constraint (4.1) in the following way

$$\begin{aligned} \sum_{i=1}^{N} g_{vi}x_i + \mathrm{UAF}_v &= 1 \quad \text{for } v = 1, \ldots, V \\ \mathrm{UAF}_v &\geq 0 \quad \text{for } v = 1, \ldots, V. \end{aligned} \tag{4.5}$$

The extra variable UAF_v for flight v is added with a very high cost coefficient Q_v in the objective function. Now it is possible to have flights not being assigned to gates, but this option comes at a high price. The unassigned flights present in the final solution are then assigned to gates manually by the gate planners, who have the ability to overrule some of the constraints, if necessary. We can select the cost coefficients of the UAF_v variables to model the effort it takes to assign a flight manually; for example, in general it is easier to assign a flight operated by a small aircraft somewhere manually than a flight operated by a big aircraft. The values for these cost coefficients were determined via preliminary computations.

One thing not addressed in the model yet is the fact that flights with a long stay can be split into three parts of which two parts must be assigned to a gate. To model this possibility, for each long-stay flight v we create two *split flights* v_A and v_B, which refer to the arrival and departure part of flight v, respectively. The intermediate parts of the flights are not modelled because the buffer stands for these intermediate parts are not part of the gate assignment problem.

Since these three flights v, v_A, and v_B concern the same aircraft, we must ensure we model their dependency. This can be achieved by splitting the original constraint (4.1) for flight v into two separate constraints

$$\sum_{i=1}^{N}(g_{vi} + g_{v_A,i})x_i + \mathrm{UAF}_{v_A} = 1 \quad \text{and} \quad \sum_{i=1}^{N}(g_{vi} + g_{v_B,i})x_i + \mathrm{UAF}_{v_B} = 1.$$

Here UAF_{v_A} and UAF_{v_B} indicate the possibility of not assigning (a part of)

flight v; their cost coefficients each get a value half of the value of the original cost coefficient UAF_v.

Furthermore, we have to include the split flights in the preference constraints 4.4. Since each split flight only represents half of the original flight, each gets a coefficient 0.5 by redefining p_{vak} as follows

$$p_{vak} = \begin{cases} 1 & \text{if flight } v \text{ has preference on gate type } a \text{ in preference } k; \\ 0.5 & \text{if the unsplit version of flight } v \text{ has preference on gate type } a \text{ in preference } k; \\ 0 & \text{otherwise} \end{cases}$$

The ILP formulation presented above models the problem correctly, but unfortunately the size of the problem is enormous, since the number of possible gate plans is enormous. Therefore we try to approximate the optimal solution by taking only *presumably useful* gate plans into account. To identify these, we first relax the integrality constraints (4.3) and then solve the resulting LP-relaxation by column generation. We then reinstate the integrality constraints and solve the resulting ILP formulation with the columns generated. As a side-effect, we can use the value of the LP-relaxation as a measure for the quality of the obtained ILP formulation.

4.2.2 Pricing problem

After we have solved the LP-relaxation for a given set of columns, we find a dual multiplier π_v for the constraint (4.1) corresponding to flight v, a dual multiplier λ_a for the constraint (4.2) corresponding to type a, and a dual multiplier ψ_k for the constraint (4.4) corresponding to preference k. Therefore, the reduced cost for a gate plan i is equal to

$$c_i - \sum_{a=1}^{A} e_{ia}\lambda_a - \sum_{v=1}^{V}(g_{vi}\pi_v + \sum_{k=1}^{K}\sum_{a=1}^{A} g_{vi}e_{ia}p_{vak}\psi_k).$$

Note that for the original parts of a long-stay flight the above must be slightly changed. Since the original part of a long-stay flight is present in two constraints, we must subtract the two dual multipliers π_{v_A} and π_{v_B} instead of only π_v.

It is well-known from the theory of linear programming that we have solved the LP-relaxation to optimality if the reduced cost of each gate plan is greater than

or equal to zero. To check this, we compute the minimum reduced cost over all feasible gate plans; this is called the *pricing problem*. We solve this problem by composing a network such that each feasible gate plan corresponds to a path in this network, and vice-versa. Moreover, we choose the lengths of the arcs such that the length of a path equals the reduced cost of the corresponding gate plan. Hence, we can then solve the pricing problem by solving a shortest-path problem in this network.

We solve the pricing problem for each gate type separately. For each type of gate a we introduce a Directed Acyclic Graph (DAG) $G_a = (V_a, E_a)$. We add a vertex to this graph for every flight v that is allowed to be assigned to a gate of type a. Furthermore, we add vertices s and t, denoting the source and sink respectively. If two flights v and w are allowed on a gate of type a and the arrival time T_w^{arr} of flight w is greater than or equal to the departure time T_v^{dep} of flight v plus the minimum idle time $T_v^{\min}$ required after flight v, then a directed edge from vertex v to w is added to the graph. Furthermore, a directed edge from the source vertex s to every vertex v is added, as well as a directed edge from every vertex v to the sink vertex t. Hence,

$$E_a = \{(v,w) | T_w^{\text{arr}} \geq T_v^{\text{dep}} + T_v^{\min}\} \cup \{(s,v),(v,t) | \text{for all } v\}.$$

It can be easily seen that every path from s to t in G_a represents a feasible gate plan of type a and vice-versa. What is left is to set the lengths of the arcs. If we look at the reduced cost, then we see that, if a flight v is selected and succeeded by flight w in a gate plan of type a, then the contribution of flight v to the reduced cost of the gate plan is

$$conv(v,w)c(T_w^{\text{arr}} - T_v^{\text{dep}}) - \pi_v - \sum_{k=1}^{K} p_{vak}\psi_k.$$

Recall that $conv(v,w)$ denotes the convenience multiplier indicating the advantage (or disadvantage) of putting flight v directly before w. We put the cost of the arc (v,w) in G_a equal to the contribution of flight v to the reduced cost. The additional arcs (s,v) and (s,t) get cost $-\lambda_a$, and the additional arcs (v,t) get cost equal to $-\pi_v - \sum_{k=1}^{K} p_{vak}\psi_k$. Note that the cost $c(T_w^{\text{dep}} - T_v^{\text{arr}})$ of putting flight w immediately after flight v in the gate plan is constant; after each iteration, we only have to update the cost terms containing the dual multipliers.

Since exactly one of the outgoing edges of vertex v will be used if v occurs in a path, the total cost of a path equals the reduced cost of the corresponding gate plan.

For solving the pricing problem we need to find the gate plan with minimal reduced cost. Since each path in the graph corresponds to a possible gate plan and the path length corresponds to the reduced cost of the represented gate plan, finding the gate plan with minimal reduced cost comes down to finding the shortest path in the presented graph. Without loss of generality we assume that all flights are sorted by their arrival times. This assumption implies a topological order on the vertices in the graph, namely the order of the flight indices. Because we now have a DAG with a topological order it is possible to find the shortest path in $\mathcal{O}(|V| + |E|)$ time (cf. Cormen et al. (2001)).

When a gate plan with minimum reduced cost has been found, there are two possibilities:

- The new gate plan has *negative reduced cost*. This means that by adding this gate plan to the master problem, the objective value of the master problem might decrease and thus we add this gate plan to the master problem.
- The gate plan has *zero reduced cost* in which case there exists no gate plan with negative reduced cost.

For each of the gate types, we need to check whether a gate plan with negative reduced cost exists. If for none of the gate types a gate plan with negative reduced cost exists, then the master problem has been solved to optimality.

4.2.3 Solving the restricted ILP

After the master problem has been solved to optimality we only have a solution for the LP-relaxation. If this solution happens to be integral, then we have a solution to the original ILP formulation of the gate assignment problem, too. If the solution is fractional, then we have to convert it to an integer solution. One possibility for this is to make use of branch-and-price. But since this is computationally infeasible, and since a good approximation suffices, we go for an approximate solution. To this end, we use the ILP-solver CPLEX to solve the ILP with the limited set of columns. In our preliminary computations, we only used the restricted set of columns generated by the column generation. It turned out that this took too much time and memory. Moreover, if solutions were

found within reasonable running time, then the quality of these solutions was really bad.The large running time and memory consumption can be explained by the fact that the restricted set of columns generated for solving the LP is too restrictive for the ILP: if flight v is part of a selected column, then none of the other columns containing v can be used anymore. Hence, if our *first-choice* column contains a flight that is included in one of the already selected columns, then we need a *second choice* column that does not contain this already covered flight. As the first-choice column was generated once, when solving a pricing problem, we decided to create a set of additional second-choice columns each time when solving the pricing problem.

To create these second-choice columns we used the following procedure. We first solve the pricing problem, that is, we find the shortest path. We then take out the nodes in the shortest path one by one and solve the shortest path problems for each of the resulting graphs. These additional columns are added to a column pool. After the master LP problem has been solved to optimality, we determine the set of unique columns from the ones that are in the column pool. This set of unique columns is then added to the restricted ILP problem. After these unique columns were added to the ILP problem, the ILP solver was able to solve the problem in a matter of minutes and sometimes even seconds instead of running for hours or even days. A second reason for this tremendous speed improvement may be that, since the number of unique gate plans that are in the column pool is quite large, it might be easier for the branch-and-bound subroutine of CPLEX to find a good lower or upper bound quickly, thus speeding up the process.

4.2.4 Assigning gate plans to gates

After solving the first phase, we have a set of gate plans with just as many gate plans of type a as there are physical gates of type a, such that the total schedule is robust against small variations caused by for example delays of flights. In the second phase of the problem we have to determine which gate plan is assigned to which physical gate.

In the first phase we have introduced constraints dealing with just one gate type. We did not consider relations between specific physical gates, like the constraint that two flights having the same departure time can not be assigned to directly opposite gates due to the impossibility of a simultaneous push-back

of both flights. We consider these types of constraints when we assign the gate plans to the physical gates in the second phase.

In Van Orden (2002) some additional constraints have been formulated that need to be addressed in the second phase. These are:

- Avoid putting two flights next to each other that have an overlapping wingspan.
- Avoid putting two flights with equal departure time next to each other because of conflicting push-back.
- Avoid putting two flights that have the same departure time on opposite gates because they cannot have a push-back at the same time.
- Flights from US carriers need extra facilities (e.g. possibility of closing parts of the waiting room at the gates to which they are assigned).
- Minimize the walking distance for the passengers. This concerns both arriving or departing passengers and transfer passengers.

Although the first one of these constraints at first sight seems to be a good example of a second phase constraint, it turned out to be not important at all. After receiving detailed information of the gates at AAS from the gate planners it turned out that this constraint did not exist for any gate at AAS. But there does exist a strongly related constraint at AAS though, which decrees that it is possible to combine two gates of a small category to one gate of a bigger category. This constraint cannot be addressed in the first phase, since we do not know then which two gate plans will be next to each other in the final solution. In theory it is possible to take this constraint into consideration when solving the first phase by creating a new category of gates consisting of the two smaller gates. The pricing problem for this gate type would then consist of two paths that both contain the selected vertices corresponding to the aircraft of a bigger category. This problem is harder to solve. Therefore, and also since there are not so many of these possibilities, we have decided not to take this constraint into consideration.

After completing the second phase it will be known which gate plans and thus which flights will be assigned to the gates that can be combined. When there

are big flights left that are still not assigned, the gate planner will be able to manually combine a set of these smaller gates into one gate of a bigger category and assign a bigger flight to this combined gate.

Although the gate planners at AAS do not have information regarding possible connecting flights of the passengers, one way they try to maximize passenger comfort is to minimize the maximum walking distance. This is achieved by putting flights with a large number of passengers at the best gates. Since we are assigning entire gate plans to gates now, we have less flexibility, but we can use the same principle. Gates that are closer to the beginning of the pier are considered to be better gates. So also in this case the number of passengers will be an important factor in the decision: when there are more gate plans to choose from, the one with the largest number of departing passengers will be assigned to the best gate.

The full assignment problem of the second phase can be decomposed into a number of smaller assignment problems for each type of gate. Most of these subproblems can be solved independently, but some are dependent, since they involve gates that have a neighboring or opposite gate of a different type. The dependent subproblems need more attention. To determine the benefit of assigning a flight to a certain physical gate, the gate planners at AAS use a number of different rules.

Presumably, the best option is to present the gate planners with the results from the first phase and have them assign the gate plans to the physical gates. This can be done *manually* since the size of these problems is rather small; generally the maximum number of gates within one gate type is around eight. Only for remote stands this number is higher, but the flexibility for assigning gate plans to these remote stands is really high. Finally, sometimes it may turn out to be beneficial to swap two flights from two gate plans, resulting in a better solution.

4.2.5 Directly assigning flights to gates

The two phases of first assigning flights to gate plans and then assigning the gate plans to gates can be integrated by modelling each single physical gate as a separate type and including all the constraints concerning specific physical gates.

As a first step, we defined in our original ILP formulation all the gates except for the remote stands as separate types. The reason we do not consider each remote stand as a separate type is that there does not exist any significant difference between these gates (e.g. they do not have waiting rooms and they all require a bus to transport passengers to and from the terminal building). Our computational experiments reveal that considering all gates as separate types is still computationally feasible. Furthermore, it decreases the amount of work in the second phase, because it limits the second phase to swapping of some gate plans between physical gates and if necessary including unassigned flights and swapping flights between gate plans.

However, the model for assigning flights to gate plans does not yet include all constraints for the case we have a separate type for each gate. The constraint that flights from the United States need extra facilities can directly be included in the definition of which flight is allowed on which type of gates. For minimizing the maximum walking distance we would need to adapt the objective function or introduce preference constraints. Moreover, preventing simultaneous push backs would also require specific constraints. To find out which constraints should preferably be included in the model and which constraints can better be handled manually by the gate planners, is a matter of further research. Presumably, there will always be need for some kind of second phase especially if it is necessary to manually violate some of the constraints to accommodate all flights.

4.3 Computational experiments

We have implemented a model and its solution in C++ and performed extensive computational experiments. All of the tests were conducted on a Pentium 4, 2.8 GHz with 1GB of RAM. For solving the LP problems and for solving the resulting ILP problem by branch-and-bound the Concert Technology interface of CPLEX 9.1.2 (ILOG, 2005) was used.

AAS has provided us with six different sets of data, three of which contain the flights on busy high season days (HS), and three on low season days (LS). From each data set we derived two instances, one instance where we group gates with equal properties into a type and one instance where, except for the remote stands, each individual gate forms a separate type. The sizes of the different data sets are given in Table 4.1. For a data set X, instance X-GG denotes the

Instance	Flights	Gate types	Total gates
HS-1-GG	699	40	128
HS-2-GG	680	40	128
HS-3-GG	688	40	128
LS-1-GG	602	40	128
LS-2-GG	608	40	128
LS-3-GG	593	40	128
HS-1-SG	699	94	128
HS-2-SG	680	94	128
HS-3-SG	688	94	128
LS-1-SG	602	94	128
LS-2-SG	608	94	128
LS-3-SG	593	94	128

Table 4.1: Sizes of the provided instances. LS is low season, HS is high season.

instance with grouped gates and X-SG the instance with a gate type for each individual gate.

During the column generation process we applied the dual simplex method for solving the LPs, which is default in CPLEX, as well as the primal simplex method. In our experiments this hardly made any difference and we report results on the dual simplex method only. To limit the size of the intermediate linear programs, we take out columns with large positive reduced cost, since these are unlikely to improve the current solution. After every given number of iterations we remove all columns from the model with reduced cost above a certain threshold. This threshold is determined by taking the average reduced cost of the columns added in the previous iteration and multiply this average with -0.75.

These removed columns are put in a special repository and are added again when we solve the ILP. We tested different frequencies of taking out columns: from every 40 iterations to every 110 iterations with step size 10, or no take out. In Table 4.2 the results are given for 40, 80, and no take out. We report on the time needed to solve the LP-relaxation, the number of iterations and the number of columns generated.

It is clear that the time needed for solving the LP is considerably bigger for the high-season data sets than for the low-season data sets. Taking out columns

	Take out freq 40			Take out freq 80			No take out		
Instance	sec.	iters.	cols.	sec.	iters.	cols.	sec.	iters.	cols.
HS-1-GG	139	693	14157	139	634	12892	195	557	12236
HS-2-GG	126	625	13622	128	580	12627	165	582	11971
HS-3-GG	110	642	13629	118	595	12628	158	537	12287
LS-1-GG	*	*	*	61	685	9910	82	640	9198
LS-2-GG	62	681	10861	67	599	9944	89	562	9640
LS-3-GG	62	693	10881	67	651	10124	82	597	9783
HS-1-SG	331	765	33004	310	652	29976	447	572	29116
HS-2-SG	300	631	31628	257	582	28410	505	589	28958
HS-3-SG	257	641	31796	236	631	28163	385	573	28423
LS-1-SG	470	1238	27975	160	700	20895	169	641	19968
LS-2-SG	137	673	24041	146	625	21960	161	562	22169
LS-3-SG	127	668	23650	122	621	22119	137	590	22276

* Take out frequency 40 resulted in an unsolvable LP

Table 4.2: Running time, number of iterations, and number of columns generated for various take-out frequency values.

every 40 iterations is faster in many cases. However, in some cases the computation time is strongly enlarged, while in one case the problem could not even be solved at all. The reason for this is that CPLEX gets stuck in a cycle of generating new columns which need some time before they become useful and before they do become useful, they are taken out and CPLEX has to regenerate them again. Our experiments indicate that a *frequency of 80 is the best* although, there is not a large difference with neighboring values.

Recall that in order to solve the resulting ILP in a reasonable time, we need to add the columns from the column pool. Moreover, we add the columns that were taken out and put in a repository. To decrease the size of the branch-and-bound tree we *gamble* for a small integrality gap. In CPLEX we manually initialize an upper bound on the best integral solution at a value of 0.35 percent above the optimal value of the LP-relaxation, which implies that all nodes with a larger lower bound are pruned. This turns out to significantly decrease the solution time. Moreover, we experimented with different settings of CPLEX, such as more elaborate preprocessing and more aggressive cut generation. In Table 4.3 we show the running times for solving the ILP for the default settings of CPLEX and for the enhanced settings. For each instance we give the average running time over the different take-out strategies and the primal and dual simplex

Instance	ILP default (s)	ILP enhanced (s)
HS-1-GG	7	6
HS-2-GG	18	9
HS-3-GG	9	8
LS-1-GG	118*	52*
LS-2-GG	101	24
LS-3-GG	58	24
HS-1-SG	21	19
HS-2-SG	48	24
HS-3-SG	26	15
LS-1-SG	168	120
LS-2-SG	73§	126
LS-3-SG	94	113

* Take out freq 40 resulted in one unsolvable LP.

§ Default CPLEX was not able to solve 3 instances within 60 minutes.

Table 4.3: Running times for solving the ILP with and without enhancements, averaged over three different values for the take-out frequency.

method for solving the LPs during column generation. In the enhanced settings, after 90 seconds, the solution process is aborted and more aggressive parameters settings are used for CPLEX, resulting in more time spent on preprocessing the problem which led to smaller running times for solving the restricted ILP. Moreover, using the enhanced settings we were able to solve more instances.

We will refer to the combination of using a take-out frequency of 80, dual simplex, and the above enhancements for the ILP as the *best variant*.

Finally, we report more details for the best variant. In Table 4.4, the running times, the number of iterations, number of generated columns, the number of columns in the column pool, together with the integrality gap are given. Here Convert denotes the time required to add all unique columns from the pool and repository to the model plus the time needed to convert the LP into an ILP. Again, a clear difference can be seen between the high season data sets and the low season data sets for both the number of generated columns and the number of columns in the column pool. The same holds for the groups of gates and single gates.

Our results indicate that the integrality gap is very small. This implies that

Instance	LP (s)	Convert (s)	ILP (s)	Total (s)
HS-1-GG	139	13	6	160
HS-2-GG	128	12	6	148
HS-3-GG	118	13	5	138
LS-1-GG	61	8	21	91
LS-2-GG	67	8	23	100
LS-3-GG	67	8	5	81
HS-1-SG	310	30	12	355
HS-2-SG	257	28	13	300
HS-3-SG	236	28	13	279
LS-1-SG	160	16	161	340
LS-2-SG	146	18	21	187
LS-3-SG	122	17	196	339

Table 4.4: Time needed for the different phases for best variant.

Instance	Iterations	Columns	Poolsize	Gap (%)
HS-1-GG	634	12892	85225	0.00
HS-2-GG	580	12627	82558	0.00
HS-3-GG	595	12628	84036	0.00
LS-1-GG	685	9910	58374	0.19
LS-2-GG	599	9944	59527	0.21
LS-3-GG	651	10124	59099	0.09
HS-1-SG	652	29976	185970	0.00
HS-2-SG	582	28410	173453	0.00
HS-3-SG	631	28163	172979	0.00
LS-1-SG	700	20895	113413	0.18
LS-2-SG	625	21960	122092	0.21
LS-3-SG	621	22119	118663	0.09

Table 4.5: Number of iterations, number of columns generated, size of the column pool, and integrality gap for best variant.

our method is able to find *practically optimal solutions* for real-life instances in about 10 minutes. Moreover, the results indicate it is feasible to consider single gates as a group. Although this does not make the second phase unnecessary, it makes it easier.

4.4 Conclusion and further research

We have investigated the gate assignment problem at AAS and developed a two phase solution approach. In the first phase we assign the flights to gate plans, while in the second phase we assign the gate plans to the actual gates. It turns out that the second phase boils down to a set of small problems that can be solved manually. For the first phase, we have presented a different way of formulating the gate assignment problem as an ILP. Our model includes all realistic constraints that are actually used for solving the gate assignment problem at AAS. Furthermore, we have described a method to find an approximate solution for this new ILP formulation. We have implemented it in C++, where we use CPLEX 9.1.2 for solving the (I)LPs.

Our implementation was tested with real life input data, provided by AAS. These instances could be solved in a matter of minutes to near-optimality and sometimes even to optimality. The planning software currently in use at AAS is based on a greedy algorithm that assigns flights to gates on the basis of an optimal point score per flight. This score includes different issues, such as preferences of airlines and ground handlers, but is does not contain any robustness measure. Our advanced LP-based algorithm and the planning software of AAS using a greedy algorithm have comparable running times. Experiences from the gate planners at AAS reveal that currently a considerable amount of time (a few hours) has to be spent on replanning the computer generated schedule in order to make it more robust, because the current planning software does not consider robustness at all. Since our algorithm focuses explicitly on robustness, the gate planners have more time for solving the conflicts that arise during the actual day. Clearly, extensive simulations and testing is required to analyse the quality of our solutions compared to the solutions presented by the current planning software.

As a first step towards the integration of the first and second phase, we defined in our first phase model every individual gate as a separate type, except for

the platform stands. Our experiments show that the problem then still can be solved within a few minutes. By increasing the number of types and defining the gate types more restrictively, we could include more constraints and preferences in the first phase. Further research possibilities are to model and implement more of the rules that the gate planners use to assign the flights to gates.

Finally, our algorithm allows parallelization, e.g. solving in parallel the pricing problems for different gate type and determining in parallel different columns for the column pool by omitting a single flight. This might be of interest when the algorithm is applied for replanning during the day of operation, because then speed is a crucial factor.