Using column generation for gate planning at Amsterdam Airport Schiphol

G. Diepen

J.M. van den Akker

J.A. Hoogeveen

J.W. Smeltink

Department of Information and Computing Sciences, Utrecht University

Technical Report UU-CS-2007-018

www.cs.uu.nl

ISSN: 0924-3275

Using column generation for gate planning at Amsterdam Airport Schiphol

G. Diepen[1]*, J.M. van den Akker[1]*, J.A. Hoogeveen[1]*, and J.W. Smeltink[2]

[1] Department for Information and Computing Sciences,
Utrecht University, P.O. Box 80089, 3508 TB Utrecht, The Netherlands.
e-mail: {diepen,marjan,slam}@cs.uu.nl
[2] National Aerospace Laboratory NLR,
Anthony Fokkerweg 2, 1059 CM Amsterdam, The Netherlands.
e-mail: smeltink@nlr.nl

Abstract. Assigning a set of flights to a set of gates at an airport used to be very easy. Due to the increased number of flights and the increased number of constraints that have to be satisfied, the assignment of flights to gates has become very complex nowadays.
In this paper we look at the gate assignment problem as it appears at Schiphol Airport Amsterdam. Given the expected arrivals and departures for the next 24 hours, we aim at finding a robust solution, such that the amount of replanning due to deviations from the expected arrival and departure times is minimum. We solve this problem by a two-phase procedure. In the first phase, we solve the problem in which we include the single-gate constraints; in the second phase, we consider the constraints in which multiple gates are involved. We solve the first-phase problem by formulating it as an integer linear program (ILP) using a completely new formulation. We find an approximately optimal solution for it by solving the ILP for a restricted set of variables. We find this restricted set by first solving the LP-relaxation problem by means of column generation. In every iteration of the column generation a small number of additional columns are also generated and put into a column pool. After the LP-relaxation problem has been solved to optimality, the generated columns and all additional columns from the column pool are used to solve the ILP problem. This part of the solution is then handed to the planners at Schiphol as an input for the second phase.
To test the performance of the algorithm we have implemented it, and we have tested this implementation with real life data provided by Amsterdam Airport Schiphol. The results are promising and the algorithm is able to solve real-life instances within acceptable running times.

1 Introduction

Between the time an aircraft lands at an airport and the time it departs again a couple of things must be done. One of the most obvious things is that the passengers need to (dis)embark the plane. Moreover, the aircraft needs to be refueled, new supplies have to be put on board, and the plane 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.

In the ideal situation, the plane lands and taxies to the gate it has been assigned to. This leads to the gate assignment problem, which is described as follows: we have to assign a set of flights to a (smaller) set of gates while making sure that certain criteria are met. Examples of such criteria are:

- Gates can handle only flights of certain sizes.
- Gates can handle only flights for certain origins/destinations.
- Gates can handle only flights of certain ground handlers.
- Two adjacent gates can not both handle big aircraft at the same time.

* Supported by BSIK grant 03018 (BRICKS: Basic Research in Informatics for Creating the Knowledge Society)

– Two adjacent gates should not handle flights with equal departure times.

Depending on the airport and its characteristics, many variants of the gate assignment problem have been researched and multiple solution methods have been suggested. A good overview of explored methods and models is given in van Orden (2002). In this paper 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 more of a capacity planning problem, and the gate planners must decide here to accept or decline new requests from airlines. The daily planning concerns creating a planning for the upcoming day on basis of the available information about the flights. Finally, the tactical planning is more like online planning, as it tries to resolve conflicts in the planned solution due to disturbances.

The kind of problem we consider in this paper is the daily planning. When looking at this planning, different objectives can be taken into account. Examples are minimizing total waiting time of aircraft or minimizing total towing actions. Another objective that is used very often in the literature is to minimize the walking distance of the passengers (Bihr 1990, Haghani and Chen 1998, Xu and Bailey 2001). Moreover, it is also possible to consider multi criteria objectives. Dorndorf, Drexl, Nikulin, and Pesch (2007) both present a state-of-the-art of the gate assignment problem in general and discuss recent developments with regard to the multi criteria objectives.

Since an airport is a very dynamic environment, the actual day will hardly ever go completely as planned; flights are subject to arrive either earlier or later than planned due to all kinds of reasons. The objective we are concerned with is to create a schedule that is able to cope with such small perturbations without having the need to replan big parts of the schedule. The number of changes should be minimal, because every change needed during the actual day has an effect on a very broad range of parties: the ground handler, security personnel, and passengers will have to go to a different location, etc. Therefore, creating a schedule that needs fewer changes during the tactical planning is beneficial to a variety of parties. We will call a schedule that leads to a minimum number of changes in the tactical planning a *robust* schedule. During meetings we had with the gate planners at AAS, finding a robust schedule was identified as the main target. The same objective has been considered by Bolat (2000) and van Orden (2002).

One problem is how to measure the robustness of a schedule. The probability that a conflict arises between two consecutive planes at the same gate depends on the 'reliability' that the planes will stick to their assumed departure and arrival times and on the time between these two consecutive flights. We cannot do anything about the first part, and hence we must divide the total available idle time, which amount is constant, optimally by optimizing the choice of the pairs of flights that we put consecutively. One approach for optimizing the individual idle times, and thus finding a robust schedule, is by minimizing the variance of the idle time between successive flights. This is the approach 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 big downside 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 paper we also aim at finding a robust schedule by trying to optimize the idle time between all consecutive flights. We do not minimize the variance, but a function based on the arc tangent, and we make some adjustments to this function to take the 'reliability' of the flights into account. We formulate this problem as an ILP, but we use a completely different representation based on the one used in Freling, Wagelmans, and Paixao (2001) for the vehicle and crew scheduling. We present an algorithm based on column generation to find a good approximation for the optimum of this model. Since the cost function is not based on actual costs, a good approximation for the optimum will suffice. Therefore, techniques as branch-and-price (Barnhart, Johnson, Nemhauser, Savelsbergh, and Vance 1998) to solve the problem to optimality are not considered.

The outline for the rest of this paper is as follows: In Section 2 we formulate the problem as an ILP and present our algorithm to find an approximate solution. Here we aggregate identical gates. In Section 3 we discuss the solving of the second phase, in which we undo the aggregation of identical gates. In Section 4 we will present the experimental results, and finally in Section 5 we draw some conclusions and indicate some future research topics.

2 First phase

2.1 Problem formulation

As mentioned in the previous section, the objective we are interested in is maximizing the robustness of a solution. To maximize the robustness we want to maximize a function of the idle times between all of the consecutive flights, ensuring that each flight can arrive either a bit too early or too late without the need for replanning the schedule.

There are several options to reward 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.

We have considered working with this objective function too, but we have decided to look for a cost function that reflects 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 arc-tangent 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. Experimental tests 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.

One of the things that must be taken into consideration during the gate assignment process at AAS is whether two consecutive flights, which are assigned to the same gate, are executed by the same airline or share the same ground handler. Such a situation is more 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 big chance that it is delayed.

To model such a relation we introduce a convenience multiplier $conv(i, j)$. To compute the cost of placing flight j immediately after flight i at the same gate, the cost corresponding to the idle time will be multiplied by this multiplier. If putting flight j after flight i is desirable (e.g. flight i and j are executed by the same airline or handled by the same ground handler), then the convenience multiplier gets a value less than 1, thus decreasing the cost. On the other hand inconvenient situations (e.g. the first flight 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 multiplication is needed.

There are several hard constraints in the gate assignment problem. Obviously, each plane must be assigned to a gate, and two planes 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 a certain group of Western European countries), 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 some exceptions. 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: flights are handled either by the KLM Ground Services, or by any other company.

For each of the gates it is also 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 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.

Since the complete gate assignment problem consists of around 600 flights and 120 gates, we would like to see if we can decompose the problem into a set of smaller sub problems based on one of the above properties. Unfortunately for each of the above properties gates exist that are multi-purpose for that particular property. This means that a strict division into multiple independent sub problems based on any of the properties is not possible. It is still possible to create a division into multiple sub problems by statically dividing the multi-purpose gates over the different sub problems, but the big disadvantage of this approach is that available capacity is discarded.

Another important feature is that some of the flights stay at AAS for a longer 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 such 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 will be towed to some buffer stand.
– Intermediate part. During this part the aircraft resides on a buffer stand, where it does not use up 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 splitting. The advantages of such splitting 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 two 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 two parts yields additional flexibility.

Currently the process of splitting the 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 resolve the problem. This step is repeated several times until sufficient capacity is available. We will incorporate the possibility of splitting the flights that remain at AAS for more than three hours as an automated option in our approach, which we describe now.

The standard approach for modelling the problem as an ILP model uses variables that denote whether a flight is assigned to a certain gate (van Orden (2002)). One major disadvantage of this formulation 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. Moreover, 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 of Amsterdam Airport Schiphol. Therefore, a new approach is needed. We split the problem in two phases and solve the first phase using a new representation, which resembles the one applied by Freling et al. (2001) for the single-depot vehicle scheduling problem. We model series of flights that are to be assigned to the same gate; we call such a series a *gate plan*. A major advantage of this new formulation over the one stated before is that feasibility can be checked easily. Furthermore, calculating the cost of a gate plan is also very easy. Now the first phase of the solving consists of finding for each group of gates with equal properties as many feasible gate plans as there are gates in that group. In the second phase the selected gate plans will be assigned to the real, physical gates matching these requirements.

For each gate plan j we introduce a decision variable x_j as follows:

$$x_j = \begin{cases} 1 \text{ if gate plan } j \text{ is selected} \\ 0 \text{ otherwise.} \end{cases}$$

Let m denote the number of flights, let H denote the number of gate types, and let n denote the number of gate plans. Now the basic model is as follows:

$$\text{Minimize} \sum_{j=1}^{n} c_j x_j \quad \text{s.t.} \tag{1}$$

$$\sum_{j=1}^{n} g_{ij} x_j = 1 \quad \text{for } i = 1, \ldots, m \tag{2}$$

$$\sum_{j=1}^{n} t_{jh} x_j = T_h \quad \text{for } h = 1, \ldots, H \tag{3}$$

$$x_j \in \{0,1\} \quad \text{for } j = 1, \ldots, n \tag{4}$$

where

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

$$t_{jh} = \begin{cases} 1 \text{ if gate plan } j \text{ is of type } h; \\ 0 \text{ otherwise.} \end{cases}$$

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. Such preferences can be used to ensure that certain flights are always assigned to the same gate due to security measures or to model that airlines have their “own” gates where at least a certain percentage of their flights have to be assigned to.

The addition of preferences can be modelled in the ILP by adding the following constraints to the model:

$$\sum_{j=1}^{n} \sum_{i=1}^{m} \sum_{h=1}^{H} p_{ihr} t_{jh} g_{ij} x_j \geq P_r \quad \text{for } r = 1, \ldots, R \tag{5}$$

where

$$p_{irh} = \begin{cases} 1 \text{ if flight } i \text{ has preference on gate type } h \text{ in preference } r; \\ 0 \text{ otherwise} \end{cases}$$

and R denotes the total number of preferences.

We further need to 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 U_i to constraint (2) in the following way

$$\sum_{j=1}^{n} g_{ij}x_j + U_i = 1 \quad \text{for } i = 1, \ldots, m \tag{6}$$
$$U_i \geq 0 \quad \text{for } i = 1, \ldots, m.$$

The extra variables are added with a very high cost coefficient 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 U_i variables to model the effort it takes to assign a flight manually; for example, in general it is easier to assign a small flight somewhere manually than a large flight.

One thing not addressed in the model yet is the fact that flights with a long stay can be split into two parts. To model this possiblity, for each long stay flight i we create two extra flights i_A and i_B, which refer to the arrival and departure part of flight i, respectively.

Since these three flights are related, we must ensure we model their dependency. This can be achieved by splitting the original constraint 2 for flight i into two separate constraints

$$\sum_{j=1}^{n}(g_{ij} + g_{i_A,j})x_j + U_{i_A} = 1 \quad \text{and} \quad \sum_{j=1}^{n}(g_{ij} + g_{i_B,j})x_j + U_{i_B} = 1.$$

Here U_{i_A} and U_{i_B} indicate the possibility of not assigning (a part of) flight i; their cost coefficients get value half of the value of the original cost coefficient U_i.

Furthermore, we have to include the extra flights in the preference constraints (5). Since the extra flights only represent half of the entire flight, they get coefficient 0.5 by redefining p_{irh} as follows

$$p_{irh} = \begin{cases} 1 & \text{if flight } i \text{ has preference on gate type } h \text{ in preference } r; \\ 0.5 & \text{if the unsplit version of flight } i \text{ has preference on gate type } h \text{ in preference } r; \\ 0 & \text{otherwise} \end{cases}$$

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 huge. 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) and then solve the resulting LP-relaxation with column generation. We use the generated gate plans and some additional gate plans that resemble the generated ones as the columns in the ILP formulation. As a side-effect, we can use the value of the LP-relaxation to measure the quality of the obtained ILP formulation.

2.2 Pricing problem

After we have solved the LP-relaxation for a given set of columns, we find a dual multiplier π_i for the constraint (2) corresponding to flight i, a dual multiplier λ_h for the constraint (3) corresponding to type h, and a dual multiplier ψ_r for the constraint (5) corresponding to preference r. Therefore, the reduced cost for a gate plan j is equal to

$$c_j - \sum_{h=1}^{H} t_{jh}\lambda_h - \sum_{i=1}^{m}(g_{ij}\pi_i + \sum_{r=1}^{R}\sum_{h=1}^{H}(g_{ij}t_{jh}p_{ihr}\psi_r)).$$

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 the path equals the reduced cost of the gate plan. Hence, we can then solve the pricing problem by solving a shortest-path in this network.

We solve the pricing problem for each gate type separately. For each type of gate h we introduce a Directed Acyclic Graph (DAG) $G_h = (V_h, E_h)$. We add a vertex to this graph for every flight i that is allowed to be assigned to a gate of type h. Furthermore, we add vertices s and t, denoting the source and sink respectively. If two flights i and j are allowed on a gate of type h and the arrival time a_j of flight j is greater than or equal to the departure time d_i of flight i plus the minimum idle time m_i required after flight i, then a directed edge from vertex i to j is added to the graph. Furthermore, a directed edge from the source vertex s to every vertex i is added, as well as a directed edge from every vertex i to the sink vertex t. Hence,

$$E_h = \{(i,j) | a_j \geq d_i + m_i\} \cup \{(s,i), (i,t) | \forall i\}.$$

It can be easily seen that every path from s to t in G_h represents a feasible gate plan of type h 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 i is selected and succeeded by flight j in a gate plan of type h, then the contribution of flight i to the reduced cost of the gate plan is

$$\text{conv}(i,j)c(a_j - d_i) - \pi_i - \sum_{r=1}^{R} p_{ihr}\psi_r.$$

Therefore, we put the cost of the arc (i,j) in G_h equal to the contribution of flight i to the reduced cost. The additional arcs (s,i) get length 0, and the additional arcs (i,t) get length equal to $-\pi_i - \sum_{r=1}^{R} p_{ihr}\psi_r$. Note that the cost of putting flight j immediately after flight i 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 i will be used if vertex i occurs in the path, the length of the path equals the reduced cost of the corresponding gate plan, except for the dual multiplier corresponding to the type of gate. We simply correct the length of the shortest path in G_h by subtracting λ_h, since we have a different graph for each gate type.

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 of the flights are sorted on their arrival times. This assumption results in 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 $\mathcal{O}(|V| + |E|)$ (Cormen, Leiserson, and Rivest 1990).

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.

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 it solves 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 there is no financial incentive involved, we decided to 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 initial 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 long running time and memory consumption could be explained by the fact that the restricted set of columns generated for solving the LP is too restrictive for the ILP: if flight i is part of a selected column, then none of the other columns containing i 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. Here we followed 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 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. A second reason that we can think of for this tremendous speed improvement is 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 more quickly and thus speeding up the process.

3 Second phase

In the first phase of the problem we have generated a gate plan for every physical gate such that the total schedule is as robust as possible against small variations caused by for example delays of flights. After solving the first phase, we have a set of gate plans with just as many gate plans of type h as there are physical gates of type h. 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 considered constraints dealing with just one independent gate or with a group of independent gates only. We did not consider relations between specific physical gates, like the constraint that two flights having the same departure time can not be put directly opposite of each other due to the impossibility of a simultaneous push-back of both flights. The reason for not addressing these constraints is that in the first phase all the gate plans are created for anonymous gates of a given type, and it is not known which gate plan will be assigned to which physical gate. We will 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 United States carriers need some extra facilities (e.g. possibility of closing parts of the waiting room at the gates they are assigned to).
– 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 could be possible to take this constraint in 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 planes of a bigger category. This problem is harder to solve, which increases the running time. Moreover, increasing the number of gate types will result in longer running times. Therefore, and also since there are not so many of these possibilities, we have decided not to take this constraint into consideration. After the second phase has finished it will be known which gate plans and thus which flights will be assigned to the gates that can be combined. When there are 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.

One of the currently important objectives is to maximize passenger comfort including minimum walking distance. This is achieved by putting flights with a large number of passengers at the best stands. 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 stands. 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 better stand.

3.1 Solving the second phase

The full assignment problem of the second phase can be decomposed into a number of smaller assignment problems for each type of gate that has been introduced. Most of these sub problems 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 sub problems need more attention.

Since we did not get all rules the gate planners use at AAS, we could only use one rule to solve the second phase. This rule states that flights having more departing passengers should be placed more at the beginning of a pier. So solving the second phase is now done by applying the following to all gate types

- Sort the gates based upon the quality (i.e. gates close by the beginning of the pier are better than gates more to the end).
- Sort the gate plans on the total number of departing passengers that are on the flights in that gate plan.
- Assign the gate plan with the highest number of departing passengers to the best gate, assign the next gate plan to the next-best gate, and so on.

It is also possible to allow more rules in the second phase. Also, sometimes it might turn out to be beneficial to swap two flights from two gate plans, resulting in a situation where a better solution for the second phase is possible. If more complex rules are applicable, another possibility is to add a local search approach that allows swapping flights within the gate plans for a certain type of gate to find better solutions. Presumably, the best option is to present the gate planner with the results from the first phase and have him assign the gate plans to the physical gates. This can still 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 the remote stands this number is higher, but the flexibility for assigning gate plans to these platforms is really high.

4 Experimental results

To test the presented algorithm we have written an implementation in C++. 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 ILP problem from the LP problem solution by branch-and-bound the Concert interface of CPLEX 9.1.2 (Ilog 2005) was used.

AAS has provided us with six different sets of data, three of which were rather busy high season days, and three low season days. The sizes of the different instances are given in Table 1.

Instance	Flights	Gate types	Gates
High-Season-1	608	43	128
High-Season-2	595	43	128
High-Season-3	592	43	128
Low-Season-1	518	43	128
Low-Season-2	530	43	128
Low-Season-3	523	43	128

Table 1. Sizes of the provided instances

In Table 2 the time needed to solve the LP-relaxation and the restricted ILP are given. It can be clearly seen that the time needed for solving the LP is considerably bigger for the high-season datasets than for the low-season datasets. For both the high season and the low season sets, there was one instance that could not be solved within 90 seconds by Cplex with the default settings. After 90 seconds, the solution process was aborted and more aggressive parameters settings were used for Cplex, resulting in more time spent on preprocessing the problem which led to acceptable running times for solving the restricted ILP.

Instance	Runtime LP-Relaxation (sec.)	Runtime ILP (sec.)
High-Season-1	161	334
High-Season-2	234	23
High-Season-3	146	5
Low-Season-1	77	261
Low-Season-2	72	13
Low-Season-3	69	12

Table 2. Running times for solving LP-Relaxation and ILP

Instance	Iterations	# Generated Columns	# Columns in column pool (*Unique*)
High-Season-1	555	13067	84284 (*56337*)
High-Season-2	559	14370	89501 (*54730*)
High-Season-3	517	13173	83004 (*53459*)
Low-Season-1	682	10874	64436 (*43744*)
Low-Season-2	594	10915	63500 (*42853*)
Low-Season-3	615	11193	64691 (*41938*)

Table 3. Generated columns.

Finally, in Table 3 the number of iterations, number of generated columns, and the number of columns in the column pool are given. Again, a clear difference can be seen between the high season datasets and the low season datasets for both the number of generated columns and the number of columns in the column pool.

5 Conclusion and further research

We have presented a new way of formulating the gate assignment problem as an ILP. This model is based on 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 formulation. We have implemented it in C++, where we use Cplex 9.1.2 for solving the (I)LP's.

The implementation was tested with real life input data, provided by AAS, and these instances could be solved in a matter of minutes to near-optimality and sometimes even to optimality. The time needed for solving the total problem for the given data is comparable to the time taken by the planning software currently in use, which is based on a greedy algorithm that assigns planes to gates on basis of an optimal point score per plane.

Further research possibilities are to try and implement more rules from what the gate planners use now.

References

Barnhart, C., Johnson, E. L., Nemhauser, G. L., Savelsbergh, M. W. P., and Vance, P. H. (1998). Branch-and-price: column generation for solving huge integer programs, *Operations Research* **46**: 316–329.

Bihr, R. A. (1990). A conceptual solution to the aircraft gate assignment problem using 0,1 linear programming, *Proceedings of the 12th annual conference on Computers and industrial engineering*, Pergamon Press, Inc., Elmsford, NY, USA, pp. 280–284.

Bolat, A. (2000). Procedures for providing robust gate assignments for arriving aircrafts, *European Journal of Operational Research* **120**: 63–80.

Cormen, T. H., Leiserson, C. E., and Rivest, R. L. (1990). *Introduction to Algorithms*, The MIT Electrical Engineering and Computer Science Series, MIT Press/McGraw Hill.

Dorndorf, U., Drexl, A., Nikulin, Y., and Pesch, E. (2007). Flight gate scheduling: State-of-the-art and recent developments, *Omega* **35**: 326–334.

Freling, R., Wagelmans, A. P. M., and Paixao, J. M. P. (2001). Models and algorithms for single-depot vehicle scheduling, *Transportation Science* **35**(2): 165–180.

Haghani, A. and Chen, M.-C. (1998). Optimizing gate assignments at airport terminals, *Transportation Research Part A: Policy and Practice* **32**: 437–454.

Ilog (2005). Ilog Cplex v9.1, http://www.ilog.fr.

van Orden, A. (2002). *Gate assignment: Methods and models*, Master's thesis, Utrecht University.

Xu, J. and Bailey, T. G. (2001). The airport gate assignment problem: Mathematical model and a tabu search algorithm., *HICSS*.