Robust Planning of Airport Platform Buses

G. Diepen

B.F.I. Pieters

J.M. van den Akker

J.A. Hoogeveen

Technical Report UU-CS-2009-026
November 2009

Department of Information and Computing Sciences
Utrecht University, Utrecht, The Netherlands
www.cs.uu.nl

ISSN: 0924-3275

Department of Information and Computing Sciences
Utrecht University
P.O. Box 80.089
3508 TB Utrecht
The Netherlands

Robust Planning of Airport Platform Buses*

G. Diepen[†§], B.F.I. Pieters [‡], J.M. van den Akker[‡], J.A. Hoogeveen,[‡]
[‡]Department for Information and Computing Sciences,
Utrecht University, P.O. Box 80089, 3508 TB Utrecht, The Netherlands
[§]Paragon Decision Technology, Schipholweg 1, 2034 LS Haarlem, The Netherlands
Guido.Diepen@aimms.com, bfipiete@cs.uu.nl, marjan@cs.uu.nl, j.a.hoogeveen@cs.uu.nl

November 27, 2009

Abstract

Most airports have two types of gates: gates with an air bridge to the terminal and remote stands. For flights at a remote stand, passengers are transported to and from the aircraft by platform buses. In this paper we investigate the problem of planning platform buses as it appears at Amsterdam Airport Schiphol. We focus on robust planning, i.e., we want to avoid that the bus planning is affected by flight delays and in this way invokes delays in other flights and ground-handling processes. We present a column generation algorithm for planning of platform buses that maximizes robustness. We also present a discrete-event simulation model to compare our algorithm to a first-come-first-served heuristic as is used in current practice. Our computational results with real-life data indicate that our algorithm significantly reduces the number of replanning steps and special recovery measures during the day of operation.

1 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. All of these actions take place while the aircraft is standing at a gate. At Amsterdam Airport Schiphol (AAS) one can distinguish two types of gates where flights can be assigned to. There are the regular gates that are equipped with a passenger air bridge, and there are the so-called

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

[†]The research was performed while this author was at Utrecht University

remote stands which do not have a bridge connection to the terminal. When an aircraft is assigned to a stand with a passenger air bridge, the passengers can (dis)embark the aircraft via this bridge. On the other hand, when an aircraft is assigned to a remote stand, the passengers will need to be transported to or from the terminal building. Since it is not an option that passengers just walk from the remote stand to the terminal building or vice versa, buses are used to transport them.

Since there are a limited number of buses available we need to determine which bus will drive from where to where at which time. This problem we will refer to as the *bus planning problem*. In this paper we consider the situation at Amsterdam Airport Schiphol. We consider the daily planning, i.e. the creation of a planning for the upcoming day as well as the tactical planning, i.e. resolving conflicts that arise in the planned solution due to disturbances during the day of operation.

We are not aware of any earlier research paper on planning of platform buses. When we look at the characteristics of this problem, one can see similarities with some other problems. One related problem is the vehicle scheduling problem of buses in public transport (see e.g. Huisman et al. (2004)). In both problems we need to determine which buses will drive what trips. A difference between the two problems is the fact that with bus planning at AAS only short trips (in the order of minutes) need to be assigned to a bus, while in the case of public transport the trips are complete lines from beginning to end and they take much longer (generally in the order of one hour or more). Furthermore, at an airport the distances are very limited and generally the time needed to drive from the end point of one trip to the start point of the next trip is very little. This means that we can be very flexible with regards to creating combinations of trips that need to be driven by one bus. This in contrast to the vehicle scheduling problem in public transport, where the time needed to drive from the end point of one trip to the start point of the next trip can be significant.

The problem also has similarities with various problems in the field of Automated Guided Vehicles (AGV). For example, the problem of routing AGV in a container terminal is related to bus planning at an airport because in both cases the goal is to assign trips to a vehicle. One major practical difference between bus planning and routing AGV is the fact that AGV have no drivers, so that we do not have to take labor regulations into account. Le-Anh and de Koster (2006) give a review of the design and control of AGV systems.

The scheduling problem of AGV is similar to the Pickup and Delivery Problem with Time Windows (PDPTW). Ropke and Cordeau (2006) proposed an algorithm for solving the PDPTW that is based on a column generation approach. Though the bus planning problem at an airport has some similarities to the PDPTW, one major difference is that at an airport, the buses must drive straight from the pickup point to the delivery point and cannot pick up more passengers in the meantime and thus the time windows are very strict. Furthermore, the same remark made earlier regarding the limitation on the distances also holds for the comparison with the PDPTW. So bus planning in this way is a special version of the PDPTW.

In previous work (Diepen (2008)),Diepen et al. (2007)), we have investigated the gate assignment problem as it appears at AAS. We included gates with an air bridge and remote

stands, which require bus transportation of passengers. We considered *robustness* as the objective function. We are concerned with creating 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. We presented a new formulation for the gate assignment problem, which enables us to compute a robust gate assignment schedule for a full day of traffic at AAS. We developed an algorithm based on column generation to find a very good approximation for the optimum of this model. Our experiments indicate that this algorithm is able to solve real-life instances to near-optimality within a few minutes.

In this paper, we focus on the bus planning problem at AAS. As with gate assignment, our objective is to maximize robustness, in the sense that the planning can cope as much as possible with flights that arrive or depart earlier and later than foreseen. Handling deviations in flight times properly is important to avoid propagation of delays into air traffic and airport processes. The delay of a flight might delay trips corresponding to another flight. If this concerns trips serving the departure of a flight, this directly results in a delayed departure. If this concerns trips serving the arrival of the flight, this implies the absence of a bus upon arrival. This may delay the cleaning and catering of the aircraft and delays the arrival of the passengers at the terminal, which on its turn invokes delays in e.g. the security process. Robust planning reduces the number of replanning operations during the day of operation. More importantly, it reduces the number of specific measures that have to be taken. An example is the last-minute deletion of a trip by reducing the number of buses serving one flight, which requires that the busplanners consult the airline.

Based on the formulation in Diepen et al. (2007), we developed an integer linear programming formulation for the bus planning problem in which we maximize robustness. We developed a column generation algorithm to solve the planning problem and performed computational experiments in which we computed a busplanning for the upcoming day based on real-life data. To further asses the robustness, we investigated the application of our algorithm as an operational planning tool. We performed a simulation study in which we compared our column generation algorithm to a first-come-first-served algorithm as it is used in current practice. We investigated the performance of both algorithms in case of disturbances in flight times during the day of operation.

The remainder of the paper is organized as follows. In Section 2 we give a problem description. In Section 3 we present the integer linear programming formulation and the column generation algorithm, and in Section 4 we report on computational experiments. After that, in Section 5 we discuss the simulation study. Finally, we conclude the paper in Section 6.

2 Problem description

As mentioned in the previous section, platform buses are used to transport the passengers to or from the aircraft standing on remote stands. For this transportation two types of buses are used at AAS. The first type of bus is called the *peak bus*, which are standard buses that are also used for city trips in public transportation. Peak buses are capable of

transporting up to 50 persons at once.

The second type of bus is the *Cobus*. It is a bus that is specifically designed for use at airports. Compared to the peak bus it has several advantages, a clear one is that it is larger and can transport up to 70 persons at once. Other advantages are that the bus has a lower floor (thus giving easy access) and it has doors on both sides. This latter advantage is particularly useful, since it will not matter from which direction an aircraft at the platform is approached.

The planners at AAS prefer to use Cobuses which are owned by the airport. The remainder of the required bus capacity is provided by peak buses delivered by the Haagse Tram Maatschappij (HTM) bus company. Every season the planners look at the planned flights and estimate the number of buses needed in each 15 minute time interval of a day for the complete upcoming season. This information is then sent to the HTM bus company, who provides the peak buses and the drivers. They will create *shifts* for buses and drivers such that they fulfill the capacity requirements for each of the 15 minute intervals. The shifts have four attributes, namely a *starting time*, an *ending time*, a *type of bus* that drives the shift, and the *number* of buses within the shift. Each bus within a shift is driven by one driver.

The planners at AAS don't have a direct influence on these shifts and they must use the shift information as input for their planning. Looking at the type of bus driving the shifts, one can see that the majority of the shifts at AAS are driven by the Cobuses.

Considering passengers, we can see that for a departure of a flight the passengers enter the bus at the bus gate at the terminal. The bus then drives to the aircraft at the platform after which the passengers disembark the bus to board the aircraft. The combination of these three events we call a *trip*. For flight arrivals one can likewise define trips consisting of these three phases, but then the origin is the aircraft and the destination is the bus gate. Furthermore, the time needed for the passengers to embark or disembark the bus is set to five minutes, independent of the number of passengers and the type of bus.

Since the number of passengers of each flight is known (the airlines should provide this to AAS), one can determine the number of trips that is needed to transport all passengers to or from the aircraft. Due to the fact that the number of passengers that can be transported with a Cobus differs from the number of passengers that can be transported with a peak bus, sometimes not all trips actually need to be served. An example of such a situation would be the transportation of 60 passengers: we either need two peak buses for this, or just one Cobus. If we create two trips but one of these trips is driven by a Cobus, the second trip is redundant because all passengers already could be transported with the one Cobus.

After running preliminary experiments we decided that in the case where there are trips for a departing flight that are possibly redundant, depending on whether the other trips for the departing flight are driven by Cobuses, we will only allow Cobuses to drive the trips for this departing flight. The reason for deciding this is that the majority of the buses that are available at AAS are Cobuses. Furthermore, there is a preference at AAS to send a Cobus as the first bus to an aircraft in case of a departing flight. This means that for departing flights only situations where the number of passengers is between 120

(one Cobus and one peak bus) and 140 (two Cobuses) would result in a possibly redundant trip because we need to send at least two buses, but possibly three. These numbers of passengers are relatively high and such flights are seldomly assigned to remote stands.

The way the trips are generated is prescribed by AAS and is different for arriving and departing flight. For an arriving aircraft, it is required that the number of buses needed to transport all passengers that will disembark are present at the moment the aircraft has come to a full stop at the remote stand.

For a departing aircraft the creation of trips is different. One major difference is that regardless of the number of passengers there must always be at least two trips for a departing flight. Furthermore, in contrast to the trips for arriving aircraft, not all trips start at the same time. The last trip must be finished with disembarking the passengers five minutes before the actual departure of the flight. The trip before the last trip must be finished with disembarking the passengers 10 minutes before the actual departure of the flight, etc.

Our goal is to find a planning for the buses that is as *robust* as possible, where robust means that having a small change in arrival or departure times of flights during the actual day itself does not imply a lot of replanning. A possible measure is to look at the *idle time* between pairs of trips that are consecutively assigned to the same bus. The idle time between two trips t and t' that are assigned consecutively to the same bus with trip t being the first trip, is defined as follows:

$$\text{idle time}(t, t') = T_{t'}^{\text{start}} - T_t^{\text{end}} - \text{driving time}(t, t'),$$

where T_t^{start} and T_t^{end} denote the start and the end time respectively of a trip t. By subtracting the "driving time" from the end location of the first trip to the start location of the second trip, we clearly obtain a nett idle time between the two trips.

To achieve a robust schedule we want to ensure that all idle times between pairs of trips that are consecutively assigned to the same bus are as big as possible. To model this within a cost function we chose to use the cost function c^B based on the one we used in Diepen et al. (2007) for the gate assignment problem. This cost function of the idle time must reflect the natural appreciation of a solution. First of all, it should penalize very small idle times with very large cost and should only mildly penalize rather large idle times. Second, the function should be steep at the beginning (for small idle times) and then flatten out, to reflect that for small idle times any improvement is very beneficial, whereas for larger idle times an extra increase is of minor importance. We found that the following cost-function based on the arctangent fulfills the desired properties best:

$$c^B(\text{idle}) = 1000(\arctan(-0.21 \times \text{idle}) + \frac{\pi}{2}), \tag{1}$$

where idle is the idle time between two trips. We did not make use of a so-called *cut-off value* (i.e. a threshold beyond which the cost will not decrease anymore if the idle time becomes bigger), since using such a cut-off value would introduce symmetry in the model. This is not preferable because it makes the resulting ILP problems more difficult to solve.

From discussions with the planners at AAS, we learned that in the case of a departing flight, there is a strong preference to let the first trip to the aircraft be driven by a Cobus. This is not a mandatory rule, but in case of choice the planners prefer this. To model this, we use an additional penalty cost in case the first trip to a departing aircraft is driven by a peak bus.

3 Column generation algorithm

In this section we describe our integer linear programming formulation and our column generation algorithm. We have multiple ways of formulating the bus planning problem as an Integer Linear Program (ILP). One standard approach is to introduce binary variables indicating whether a bus serves one trip after another trip (this order is needed to calculate the cost because it is based on the time between two trips) but, as for gate assignment, this approach would result in a model that is way too large to handle.

To cope with this problem we will use an ILP formulation similar to the formulation we used in Diepen et al. (2007), for the gate assignment problem. We introduce *bus plans*, which consist of a set of trips that will be driven by one bus. Bus plans correspond to shifts. For a given bus plan to be feasible we require that:

- all trips present in the bus plan can be realized within the start and end times of the associated shift;
- all trips within the bus plan are allowed to be performed by the type of bus of the associated shift;
- no two trips are conflicting (i.e. there must be enough time to drive from the destination of one trip to the origin of the next trip).

For the shifts that last longer than 4.5 hours (so-called *long shifts*), we need an extra constraint on the set of trips being driven. Whenever a driver has a shift that lasts more than 4.5 hours, he is required by law to have a 45 minute break that does not start within 1.5 hours of the shift start time and does not end within 1.5 hours of the shift end time. So somewhere around the middle of the shift, a 45 minute period of time should be reserved for a break.

One problem that is introduced by arriving flights with many passengers is that several trips are needed, all with exactly the same origin and destination. This results in symmetry in the model that we would rather not have. To resolve this situation we introduce so-called *supertrips*. A supertrip is like a normal trip, except that instead of having to be driven by exactly one bus, it must be driven by q buses, where q is determined by the number of passengers to be transported. So instead of having q normal trips that all need to be driven once, we now have one supertrip that needs to be driven exactly q times.

We formulate the bus planning problem as an ILP that uses this notion of bus plans in the following way. Suppose that we are given the complete set of all possible bus plans. To solve the problem, we have to select a subset from this complete set such that each trip

is present in exactly one of the selected bus plans (and super trips are present in q bus plans). We must ensure that we have as many bus plans selected for a given shift as there are buses within that shift. Furthermore, we can determine the cost c_j of a bus plan j by summing the cost of all successive pairs of trips in the bus plan. In a plan for a peak bus we add penalty cost for each trip that is the first trip for a departure. The objective now is to minimize the total cost of the set of the selected bus plans.

Introduce a binary variable y_j for each bus plan j as follows:

$$y_j = \begin{cases} 1 & \text{if bus plan } j \text{ is selected;} \\ 0 & \text{otherwise.} \end{cases}$$

Now the model is as follows:

$$\text{Minimize } \sum_{j=1}^{M} c_j y_j + \sum_{t=1}^{T} R_t \text{UAT}_t$$

subject to:

$$\text{UAT}_t + \sum_{j=1}^{M} h_{tj} y_j = S_t \quad \text{for } t = 1, \ldots, T \tag{2}$$

$$\sum_{j=1}^{M} f_{jb} y_j = T_b \quad \text{for } b = 1, \ldots, B \tag{3}$$

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

Constraint (2) ensures each trip is driven the correct number of times. In this constraint

$$h_{tj} = \begin{cases} 1 & \text{if trip } t \text{ is present in bus plan } j; \\ 0 & \text{otherwise,} \end{cases}$$

M denotes the total number of available bus plans, and S_t denotes the number of times trip t needs to be covered, which is equal to one for all normal trips and for a supertrip it is equal to the number of trips it is composed of. Finally, UAT_t (unassigned trip) allows for trip t being unassigned in case not enough buses are present to drive all trips at a certain time. To ensure trips are assigned whenever enough buses are present, the UAT_t variable for each trip t gets a very large cost coefficient R_t in the objective function. The value of this cost coefficient has been determined through preliminary experiments.

Constraint (3) ensures we select as many bus plans for each shift as there are buses present for that shift. Here we have

$$f_{jb} = \begin{cases} 1 & \text{if bus plan } j \text{ corresponds to shift } b; \\ 0 & \text{otherwise,} \end{cases}$$

and T_b denotes the number of buses shift b consists of.

3.1 Pricing problem

We solve the model for the bus planning problem by means of column generation. Since the total number of possible bus plans is enormous, we work with a small subset of the possible columns. For this subset we solve the above model, which gives us a dual multiplier ϕ_t for the constraint (2) corresponding to trip t and a dual multiplier ω_b for the constraint (3) corresponding to shift b. The reduced cost of a bus plan j for shift b is equal to

$$c_j - \sum_{t=1}^{T} h_{tj}\phi_t - \omega_b.$$

After solving the above model with the subset of the bus plans, we need to determine whether there exists a bus plan with negative reduced costs. It is not possible to check the reduced cost of all bus plans, again because the total number of possible bus plans is enormous. In the pricing problem we want to find the bus plan with minimum reduced cost. As soon as we have found the minimum there are two options: the bus plan has negative reduced cost, meaning that adding this bus plan to the model might decrease the objective value. Or, the bus plan has reduced cost ≥ 0, meaning that adding this bus plan will not decrease the objective value and thus the optimum is found.

We solve the pricing problem for each shift b separately. We introduce a Directed Acyclic Graph (DAG) $G_b = (V, E)$ for every shift b. We add a vertex corresponding to trip t to G_b if trip t is within the start and end time of shift b and can be performed by the type of bus of shift b. We add an edge between two vertices if the two corresponding trips are not conflicting. Now a path through the graph G_b represents a feasible bus plan for shift b and vice versa, all feasible bus plans for shift b can be represented by a path through the graph G_b.

The above only holds for shifts which do not require a mandatory break. In case we have a long shift, we have to provide a way of putting a break somewhere in the middle of the shift. We can do this by adding *break vertices* to the graphs corresponding to shifts which last longer than 4.5 hours in the following way:

- Add a break vertex between two trips, such that between the trips there is time for a break of at least 45 minutes and the driving time to and from the break location;
- Add a break vertex between the source vertex and any vertex of which the corresponding trip has a start time that is at least 135 minutes (i.e. 90 minutes buffer and 45 minutes break) later than the start time of the shift. This indicates a driver starts his shift doing nothing and then starts his break;
- Add a break vertex between the sink vertex and any vertex of which the corresponding trip has an end time that is least 135 minutes earlier than the end time of the shift. This indicates a driver has his break and after that does not drive any trips anymore for the duration of his shift.

Finally, we also add the extreme situation in which a driver starts the shift, and is not assigned to any trip for the complete duration of the shift and only is assigned to have a

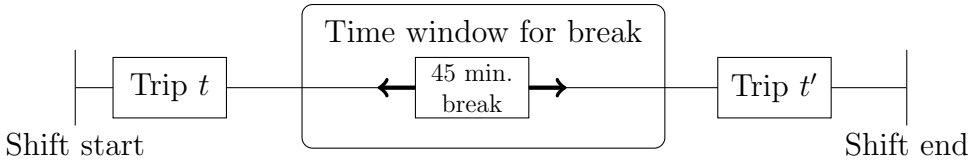

Figure 1: A time window within which a break must be held.

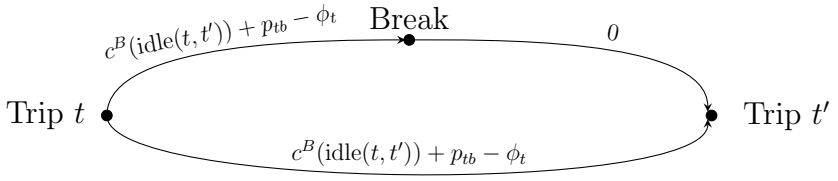

Figure 2: Example of possible break between two trips.

break within the given limits. This is accomplished by adding a break vertex between the source and sink vertex, with an edge from the source to the break and one from the break to the sink.

With these break vertices added we can represent any feasible bus plan for a long shift b with a path through the graph G_b. However, not every path through the graph corresponds to a feasible bus plan, since a feasible bus plan has to contain exactly one break vertex. We will address this issue while solving the pricing problem.

One note to be made regarding the breaks is that with the above approach we do not plan the exact start time and end time of a break but we give a *time window* in which the break must be held. An example of this time window can be seen in Figure 1 where the 45 minute break must take place somewhere in the depicted time window.

We now want to put cost on each of the edges in such a way that a path through the graph does not only correspond to a feasible bus plan, but also that the total cost of the path is equal to the reduced cost of the corresponding bus plan. When looking at the reduced cost of a bus plan, the contribution of trip t if it is followed by trip t' is exactly:

$$c^B(\text{idle}(t,t')) + p_{tb} - \phi_t,$$

Here p_{tb} are the penalty cost for assigning trip t to shift type b if t is the first trip for a departing flight and b is the shift of a peak bus. The value p_{tb} is zero otherwise. Furhtermore, ϕ_t is the dual multiplier corresponding to the trip constraint (3) for trip t. In case of a possible break between trips t and t', denoted by the presence of a secondary path from t to t' through a break vertex, the cost of the edge from vertex t to the break vertex will be equal to the cost of the edge from vertex t to vertex t'. The edge from the break vertex to vertex t' will have cost 0. This cost distribution is depicted in Figure 2.

Furthermore, the cost of all outgoing edges from the source vertex will get cost $-\omega_b$, where ω_b is the dual multiplier corresponding to the constraint (3) for shift b.

In case of a shift without a break, a path through the graph corresponds to a feasible bus plan and the total cost of the path is equal to the reduced cost of the bus plan. In case of a shift that needs a mandatory break, any path through the graph that contains exactly one break vertex also corresponds to a feasible bus plan and again the total cost of the path is equal to the reduced cost of the bus plan. Finding the bus plan with minimum reduced cost for a given shift b now comes down to finding the shortest source-to-sink path in the graph G_b. For the shifts that have a duration of less than 4.5 hours, solving the shortest path problem is straight-forward. Since we have a DAG with topological order this can be done in $\mathcal{O}(V + E)$ time (cf. Cormen et al. (2001)). For the shifts that have a duration longer than 4.5 hours, and therefore require a mandatory break, determining the bus plan with minimum reduced cost takes a bit more work. Instead of having a dynamic programming problem with only one state variable per vertex for holding the cost of the minimum cost path to this vertex, we also need a second state variable for holding the cost of the minimum cost path with a break to this vertex. Furthermore, we also need to keep two references to the predecessors: one for the case a break has been taken already and one for the case no break has been taken.

Additionally, we must make two minor modifications to the behavior of updating the cost of the successor vertices in the dynamic programming algorithm:

- Do not update the minimum cost value with a break of any successor that is a break vertex because it is not possible to have more than one break vertex in a path.
- When the current vertex is a break vertex, only update the minimum cost value with a break of the successors with the minimum cost value without a break of the current vertex. Recall from Figure 2 that the cost of the outgoing edges of a break vertex is set to 0.

The required modifications will not change the complexity of the algorithm because we only add a constant number of additional operations per vertex. So the running time of the algorithm for finding the shortest path with breaks also is $\mathcal{O}(V + E)$.

If our shortest path algorithm cannot find a new bus plan with negative reduced costs for any of the bus shifts, the value of the LP-relaxation cannot be improved anymore and this means that the LP-relaxation has been solved to optimality.

3.2 Solving the ILP

Since the solution to the LP-relaxation might be fractional, we have to ensure that we end up with an integral solution. We achieve this by reinserting the integrality constraints and solving the problem with, among others, the set of columns generated during the column generation as an ILP. It turns out that using only the set of columns generated during the column generation is often too restrictive for the ILP, in the sense that this results in quite bad integral solutions, which come at the expense of big running times.

We resolve this difficulty by creating a set of extra columns during the column generation process. These extra columns are put in a column pool and are generated in the following way:

- For each trip-vertex belonging to the path with minimum cost:
  - Take the vertex out of the graph
  - Solve the shortest path problem again
  - Put the bus plan corresponding to the new shortest path in the column pool
  - Reinsert the vertex again in the graph.

Note that when solving the shortest path after one of the vertices has been removed, we do not require the total cost of the shortest path to be negative (i.e. we do not require negative reduced cost for these additional columns).

By generating the columns this way they are not completely random, but quite related to the columns needed for solving the LP-relaxation to optimality.

After the column generation is finished, all unique columns from the column pool are added as candidates to the ILP-formulation of the problem. After these columns have been added, the problem has become considerably larger in number of variables. Fortunately, the set of columns is now less restrictive, enabling the solver to find feasible, good integral solutions easier, which usually are better than the solution of the problem without additional columns.

Our preliminary experiments showed that the integrality gap between the solution value of the LP-relaxation and the final integral solution was really small. As a consequence we can solve the ILP in two stages. First we assume a small integrality gap by supplying an upper bound to the solver which will prune all nodes with a relaxation value above it (i.e. it behaves as if an integral solution with the given value has been found already). Because of this tight upper bound, we can also have the solver put more emphasis on finding integral solutions, instead of proving the optimum. Setting the emphasis on integral solutions will result in different branching schemes and node selections being used by the solver.

If this tight upper bound leads to an infeasible ILP problem, we will remove the upper bound and solve the problem again. After preliminary experiments, the upper bound was set such that for all instances the ILP problem was still feasible.

4 Computational experiments

For testing our solution method for the bus planning problem we have implemented the algorithm in C++. The computer on which we ran the experiments was equipped with an Intel Pentium 4 3.00 GHz processor and 1 GB of RAM. For solving the LPs and ILPs we made use of the Concert Technology interface to CPLEX 9.1.3 (ILOG, 2005).

For testing our algorithm, AAS provided us with all data regarding buses and flights for one complete month. In Table 1 the sizes of the instances provided by AAS are given. The column with the number of supertrips gives the number out of the total number of trips which represent supertrips (i.e. trips that need to be covered more than once). In the column with the number of trips the supertrips are counted as one trip. Almost all of the supertrips consist of trips that need to be covered just twice.

Instance	# Flights	# trips	# supertrips	# Shifts	# buses
1	198	299	49	14	50
2	230	346	41	21	63
3	248	372	41	23	61
4	252	379	31	24	64
5	253	380	31	19	64
6	202	305	17	23	62
7	227	343	43	20	62
8	192	292	25	28	113
9	261	391	37	24	61
10	249	378	23	24	64
11	250	378	28	21	64
12	249	377	35	26	62
13	254	382	54	21	62
14	199	303	36	13	50
15	222	332	32	21	63
16	251	378	40	22	61
17	260	390	21	24	64
18	253	382	29	23	64
19	263	394	35	23	62
20	253	382	58	20	62
21	192	290	50	14	50
22	214	318	42	21	63
23	245	369	41	26	61
24	230	344	16	25	64
25	250	373	22	21	64
26	252	380	33	24	62
27	257	387	43	22	62
28	192	287	38	15	50
29	199	302	24	20	63
30	228	342	27	26	61

Table 1: Instance sizes.

The effect of the improvements separately

To avoid symmetry in the model, we introduced supertrips. To investigate the effect of using supertrips, we solved the instances not only with supertrips enabled, but also with supertrips disabled (i.e. every supertrip t is replaced by S_t identical separate trips). Furthermore, we also investigate the effect of *column deletion* (CD), which is removing columns from the model every given number of iterations. We experimented with different values for the parameters of the column deletion and based on these experiments we used the following values for the parameters: every 30 iterations we determine the average reduced cost of the columns added in the previous iteration and remove all columns from the model that have reduced cost larger than -0.75 times this average.

For supertrips disabled and enabled, we solved the problem both without and with column deletion. In Table 2 and Table 3 we present for all cases the number of iterations it took for the column generation to find the optimum, as well as the number of columns that were present in the model when the column generation was finished.

Our experiments indidate that the effect of using column deletion on the number of iterations needed goes both ways: there are some instances for which the use of column deletion decreases the total number of iterations needed, but there are also instances where the use of column deletion increases the number of iterations.

The use of column deletion does yield a significant reduction in the number of variables

Instance	Iterations		Columns in model		Time (s) for LP	
	No CD	With CD	No CD	With CD	No CD	With CD
1	568	621	6023	1444	103.6	79.5
2	459	428	7664	1739	172.4	106.0
3	534	559	8975	1872	241.7	158.2
4	549	598	9279	2220	246.2	163.2
5	610	684	8794	1956	236.2	160.8
6	306	301	5431	1152	51.6	38.2
7	525	560	7609	1804	168.6	111.2
8	375	420	7833	2604	68.0	63.9
10	584	570	9431	2091	292.5	173.5
11	469	489	8780	1816	214.4	143.2
12	594	563	9321	1884	311.0	139.6
13	535	486	9008	1883	319.5	145.7
14	685	635	9671	2007	439.4	196.3
15	663	709	6178	1406	97.6	72.2
16	345	373	6041	1434	100.1	76.9
17	542	538	8391	1927	212.2	155.1
18	630	589	10243	1900	305.0	167.6
19	475	511	9020	2128	246.1	152.3
20	529	640	9157	1749	304.1	206.2
21	586	585	8842	1808	301.8	167.7
22	568	602	5778	1359	79.1	61.2
23	421	425	7296	1531	122.8	79.8
24	509	518	9701	1913	318.2	157.6
25	449	478	8157	1830	145.0	87.3
26	578	536	9272	1933	267.2	132.4
27	527	544	9375	1900	281.6	164.3
28	670	738	9637	1926	575.9	237.9
29	586	606	6246	1503	88.2	62.7
30	435	456	6462	1708	81.0	56.6
31	427	418	7763	1797	130.7	85.2

Table 2: Overview results LP without supertrips, both without and with column deletion.

present in the model when the optimum is reached; reductions up to 80% can be seen. This massive reduction results in a shorter time needed for re-solving the restricted master problem after adding the new columns in each iteration of the column generation process. The fact that the models in each iteration are smaller in size, meaning they take less time to be solved, results in the decrease in running times as seen in Table 2 and Table 3. Even in the case where more column generation iterations are needed to solve the LP-relaxation, the total time needed for solving the LP-relaxation decreases due to the size of the models in each iteration.

When we combine the results of Table 2 and Table 3, we see that the combination of using column deletion and using supertrips yields the best result. The minimum improvement encountered is a speedup of a factor 1.49, while the average speedup is 2.39. Furthermore, when looking at the speedup achieved compared to the size of the running time of the original problem, we can see that there is a trend of the speedup increasing with increasing running times of the original problems.

Default CPLEX versus CPLEX with improvements

In Table 4 we present information regarding the resulting ILP problems. We compare the time needed for solving the ILP both without and with the upper bound obtained by

Instance	Iterations		Columns in model		Time (s) for LP	
	No CD	With CD	No CD	With CD	No CD	With CD
1	523	586	5325	1050	57.6	49.2
2	377	404	6334	1521	90.9	69.8
3	501	481	8187	1648	175.2	99.0
4	521	518	8779	1959	171.0	111.5
5	533	545	7752	1787	152.0	105.4
6	271	329	4927	1136	37.3	32.5
7	482	490	6834	1438	107.1	70.1
8	360	371	7317	2157	49.9	45.8
10	527	523	8352	1778	208.9	119.4
11	411	483	7794	1807	148.7	114.9
12	528	530	8353	1788	182.3	111.3
13	516	463	8127	1678	300.2	109.4
14	588	577	8163	1533	302.7	116.2
15	639	655	5750	1121	68.6	48.6
16	336	377	5600	1273	69.6	58.6
17	465	516	7458	1701	145.6	107.4
18	604	564	9771	1700	266.0	140.3
19	445	462	8135	1873	170.8	116.1
20	508	509	8363	1670	211.0	127.2
21	500	539	7624	1487	171.7	101.1
22	483	514	4843	1100	43.1	33.9
23	364	367	6149	1251	67.1	50.4
24	476	470	8923	1577	228.0	103.2
25	446	461	7937	1615	125.8	71.3
26	529	532	8620	1851	257.3	109.3
27	524	493	8887	1584	237.1	121.4
28	581	599	8317	1573	283.3	123.2
29	512	554	5453	1295	53.0	41.3
30	419	426	6115	1375	61.2	42.4
31	409	372	7337	1568	98.2	62.9

Table 3: Overview results LP with supertrips, both with and without column deletion.

assuming a small integrality gap. We can see that the upper bound enables us to solve all the ILP models within two minutes. Without using the tight upper bound approach two of the instances are not solvable within a set time-limit of one hour.

Looking at the instances that are solved by both the default and the enhanced settings, we can see that the time needed by the default settings often is less than the time needed when the enhanced settings are used. This can be explained by the fact that with the enhanced settings a considerable amount of time is spent by CPLEX on pre-processing the root-node of the branch-and-bound tree. Due to the aggressive settings supplied to CPLEX for cut-generation, solving the LP-relaxation of the root-node and adding the cuts takes quite some time.

After the rootnode has been solved, the settings for cut-generation are set to default again. Furthermore, we set the emphasis for the solver to finding integral solutions because all integral solutions will fall within the given tight integrality gap.

In Table 5 we present all information regarding the best settings for solving the problem. We can see from the table that the time needed for solving the pricing problems is about 50% of the total time needed for solving the LP-relaxation.

In the time needed for solving the pricing problems also the time needed for creating the additional bus plans for the column pool is incorporated. The creation of these additional bus plans accounts for roughly 70% of the time needed for solving the pricing problems.

Instance	Number of rows	Number of columns	Percentage Non-Zero	Time (s) ILP Default	Time (s) ILP Enhanced
1	313	40615	3.38	51.9	31.1
2	367	49384	2.98	5.6	13.0
3	395	58404	2.68	70.3	32.3
4	403	67855	2.65	5.4	17.9
5	399	58095	2.77	6.5	10.5
6	328	36047	3.1	1.2	3.9
7	363	51173	3.03	3.5	8.4
8	320	46378	2.69	1.8	4.6
10	415	62555	2.63	8.6	22.9
11	402	65530	2.69	*	54.7
12	399	64603	2.76	4.6	11.5
13	403	61269	2.73	5.1	9.2
14	403	61517	2.76	4.3	9.0
15	316	41331	3.19	3.4	9.4
16	353	47087	3	*	101.7
17	400	57743	2.7	39.5	35.5
18	414	71978	2.64	12.0	27.5
19	405	68003	2.77	7.7	18.7
20	417	65201	2.69	4.7	11.1
21	402	58176	2.79	2.6	6.2
22	304	33664	3.21	4.0	15.4
23	339	44892	3.06	3.1	5.2
24	395	65459	2.67	6.3	9.7
25	369	54702	2.73	2.2	5.9
26	394	65128	2.75	6.5	13.1
27	404	67885	2.71	5.0	9.7
28	409	62702	2.71	6.2	14.7
29	302	38544	3.25	2.3	7.0
30	322	42764	3	2.9	6.8
31	368	49768	2.69	2.7	6.5

*: Not Solvable within one hour

Table 4: The ILP information for both the default settings and enhanced settings with a tight upper bound.

Due to its structure the creation of these additional columns could be separated over multiple computers or threads because their creation only depends on the graph and a set of dual multipliers.

Since in most cases the time needed for solving the LP-relaxation is larger than the time needed for solving the resulting ILP, a considerable improvement of the total running times can be achieved by parallelizing the solving of the pricing problems and also the creation of the additional columns.

To investigate the effect on the time needed for solving both the LP-relaxation and the ILP, we plot the respective running times against the number of trips.

In Figure 3 we have plotted the time needed for solving the LP-relaxation against the number of trips. We can see that the use of supertrips only incurs an improvement on the solution time for the LP-relaxation due to the decreased number of trips in the model. It can be seen that the solution times needed when solving the LP-relaxation without supertrips are in line with what would be expected when solving problem with larger amount of trips.

When we take a look at Figure 4, where we have plotted the time needed for solving the

Instance	Number of iterations	Size of pool	Columns taken out	Time (s) needed for solving Pricing	LP	ILP	Total
1	586	50189	4800	25.9	49.2	31.1	87.2
2	404	59287	5205	37.8	69.8	13.0	93.3
3	481	68237	6631	44.7	99.0	32.3	146.2
4	518	79710	7465	58.6	111.5	17.9	146.3
5	545	71700	6376	56.3	105.4	10.5	131.1
6	329	41757	4233	17.1	32.5	3.9	43.7
7	490	62626	5734	37.7	70.1	8.4	91.0
8	371	45355	5746	26.5	45.8	4.6	59.0
10	523	75863	6963	58.2	119.4	22.9	158.9
11	483	75460	7190	58.3	114.9	54.7	186.0
12	530	77342	6996	60.3	111.3	11.5	138.8
13	463	71605	6586	53.9	109.4	9.2	135.1
14	577	75736	6897	57.9	116.2	9.0	141.2
15	655	51075	5161	25.1	48.6	9.4	65.0
16	377	53986	5220	30.7	58.6	101.7	169.8
17	516	70275	6529	53.7	107.4	35.5	158.2
18	564	85648	8189	71.0	140.3	27.5	185.9
19	462	79072	6877	59.6	116.1	18.7	151.8
20	509	78166	7110	67.0	127.2	11.1	156.6
21	539	72740	6441	54.6	101.1	6.2	123.1
22	514	42103	4168	18.4	33.9	15.4	55.4
23	367	52971	5177	26.5	50.4	5.2	64.5
24	470	78872	7729	47.3	103.2	9.7	129.2
25	461	64228	6808	37.8	71.3	5.9	90.3
26	532	76408	6978	56.0	109.3	13.1	138.6
27	493	78797	7530	61.2	121.4	9.7	148.0
28	599	76563	6946	60.4	123.2	14.7	154.5
29	554	47765	4695	20.3	41.3	7.0	54.5
30	426	47560	4954	21.4	42.4	6.8	57.2
31	372	56164	5847	29.4	62.9	6.5	81.6

Table 5: All information with regards to the optimal approach.

ILP against the number of trips, we can see that the time needed for solving the resulting ILP behaves a bit more erratic compared to the running times needed for the LP. We can see though that generally the time needed for solving the ILP is influenced in a positive way when the symmetry is removed by using the supertrips.

The effect of using supertrips simplifies the problem in two ways: first of all the symmetry is removed, secondly the number of trips is decreased. While solving the LP-relaxation benefits more from the decrease in the number of trips, solving the resulting ILP benefits more from the fact that the symmetry is removed.

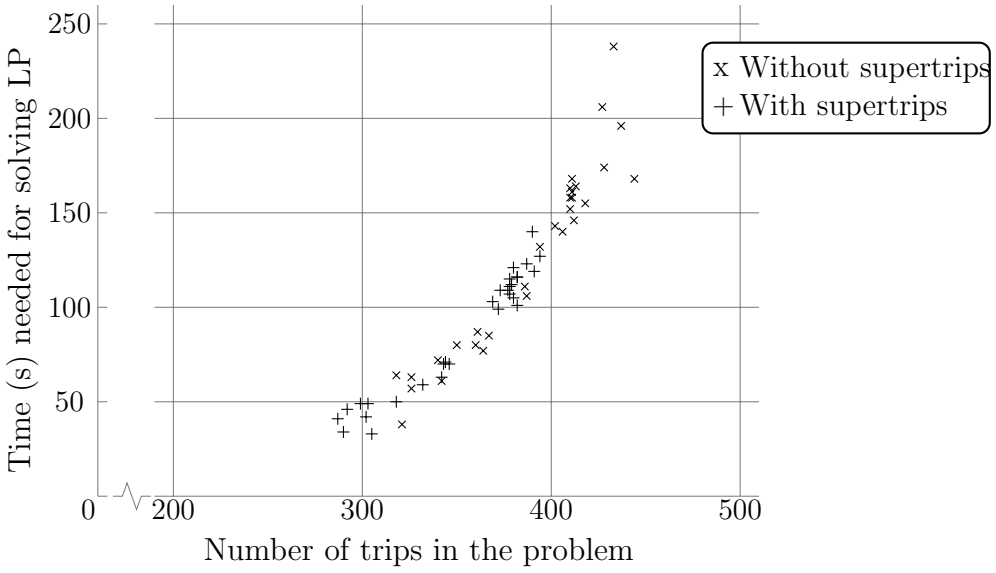

Figure 3: Time needed to solve the LP-relaxation in relation to number of trips.

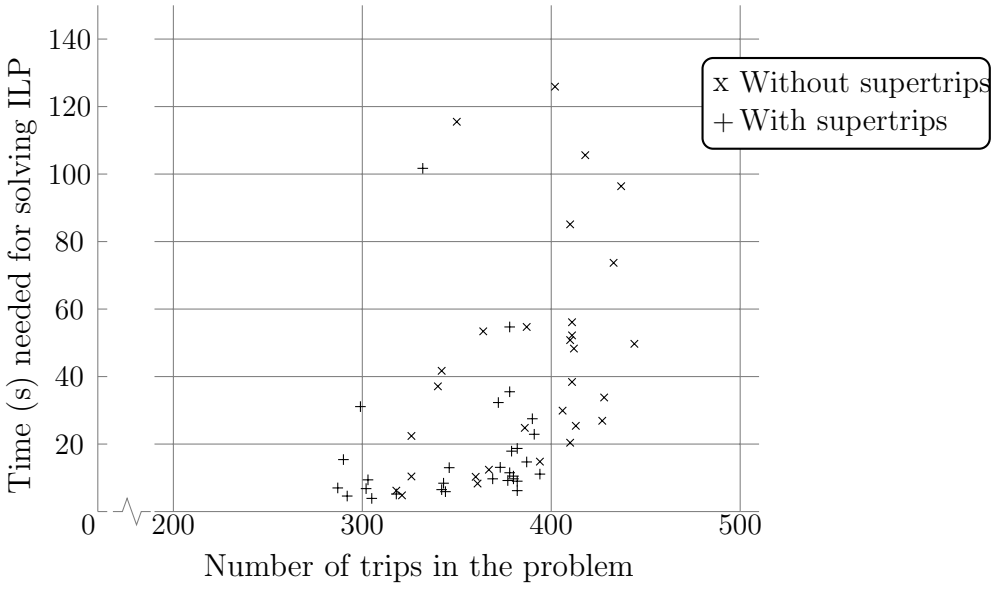

Figure 4: Time needed to solve the ILP in relation to number of trips.

5 Simulation experiments

Our column generation algorithm is designed to maximize robustness. In this section we present a simulation study to assess the performance of this algorithm in case of disturbances in the departure and arrival times of flights during the day of operation. In the study, we compare column generation with the algorithm that is used in current practice. From conversations with the planners at AAS and observations of the scheduling tool at AAS, it became clear that the current system uses a first-come-first-served algorithm. Recall that depending on the number of passengers an arrival or departure requires a given number of trips that must be performed by a Cobus (capacity 70) and a given number of trips that can also be performed by a peak bus (capacity 50). The planners at AAS assign a Cobus to a trip whenever a Cobus is available even if this is not strictly required. We model current practice by the following algorithm which we call FCFS-CP:

1. Assign trips to buses in order of increasing start time.
2. If a Cobus is available at the starting time of a trip, assign the trip to the Cobus that has been idle for the longest time.
3. Otherwise, if the trip can be performed by a peak bus and a peak bus is available, assign a trip to the peak bus which has been idle for the longest time.
4. Otherwise, the trip is marked as unassigned.

For the sake of completeness, in our experiments we included the same FCFS algorithm called FCFS-NP, without preference for a Cobus. In this algorithm Step 2 and 3 are replaced by the following. A trip that has to be performed by a Cobus is assigned to the Cobus that has been idle for the longest time at the starting time of the trip, provided that a Cobus is available then. A trip that can also be performed by a peak bus is assigned to the bus (Cobus or peak bus) that has been idle for the longest time at the starting time of the trip, unless no bus is available then.

Figures 5 and 6 show part of a schedule computed by column generation and by FCFS respectively for the same instance. This illustrates that column generation spreads trips more evenly. Moreover, the column generation algorithm will end up with fewer unassigned trips, since unassigned trips have large cost in the objective function.

In Table 6 we compare the robustness values of the initial schedules. For each algorithm we give the value of the cost of non-robustness,i.e., the total values of c^B(idle) defined in (1) in Section 2. and the ratio of colgen and the FCFS algorithms. Moreover, we include the number of unassigned trips. We can see that the column generation algorithm results in significantly fewer unassigned trips. Compared to FCFS-CP reductions of the cost of non-robustness of about 50 percent are observed. FCFS-NP sometimes results in lower non-robustness cost, but this comes at the expense of many more unassigned trips.

At Schiphol all flight information is maintained in the Central Information System Schiphol (CISS). During the day of operation, flights arrive and depart either earlier or later than foreseen due to all kinds of reasons. This means that the data in CISS are

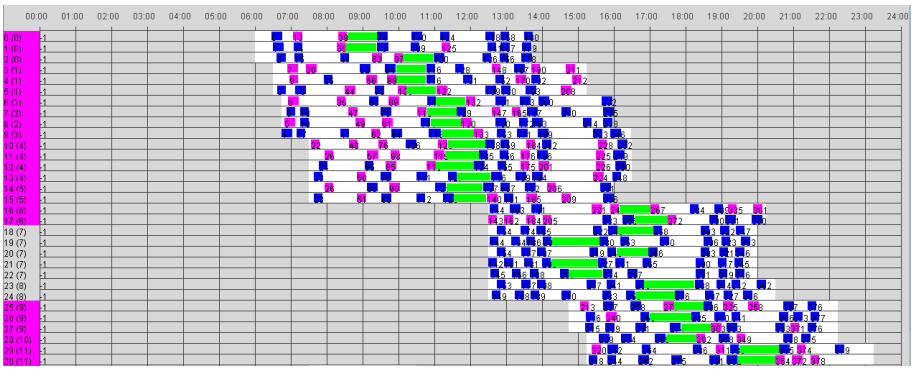

Figure 5: Part of a schedule calculated by column generation

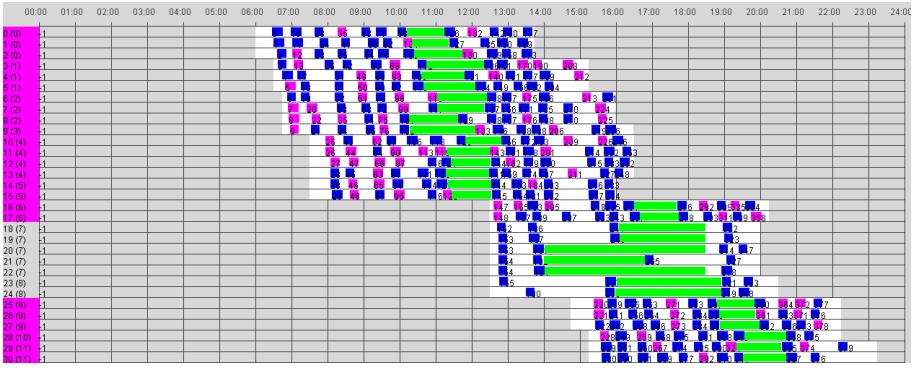

Figure 6: Part of a schedule calculated by FCFS-CP

updated over and over again. As a consequence, the planning of the platform buses also has to be updated. In our simulation we follow the bus planning and its updates throughout the day of operation.

To model the arrival times of updates in the flight times and the size of the disturbances in arrival and departure times of flights we analyzed flight information date from CISS for seven subsequent days. Although flight times can be updated a number of times, in the simulation, we restrict ourselves to the last update. The data roughly show that the time between the last update and the updated flight time follows an exponential distribution. We set the mean to 15 minutes. We made density plots for the difference between the scheduled flight time and the last updated flight times for arriving as well as departing flights. These plots indicate that for arriving flights this difference follows a normal distribution. For departing flights, it can be modeled by a composition of the exponential distribution with one part for early departures and one part for late departures.

In the following we describe the simulation procedure. The scheduler can either be the column generation algorithm or one of the FCFS algorithms, depending on the algorithm that is tested. The simulation is initialized at the beginning of the day. The scheduler

Instance	Non-Robustness			Ratio Non-Robustness		Unassigned trips		
	CG	FCFS-CP	FCFS-NP	CG/FCFS-CP	CG/FCFS-NP	CG	FCFS-CP	FCFS-NP
Day 5	130055	273400	161062	0.476	0.807	5	17	8
Day 15	173738	242027	122929	0.718	1.413	4	17	12
Day 20	188171	316173	209771	0.595	0.897	6	31	22
Day 25	71045	260228	96449	0.273	0.737	2	13	8
Day 30	57318	165312	64280	0.347	0.892	2	10	8
avg ratio				0.482	0.949			

Table 6: Robustness and unassigned trips in initial schedules.

then makes a complete plan for the upcoming day.

Every update message for a flight arrival or departure time invokes an event in the simulation. Upon arrival of an update message, the trips corresponding to the updated flight are moved in the planning. If the update results in an overlap of two trips or in a swap of the order of two trips in a bus plan, we say that there is a conflict. In this case the scheduler computes a new planning on the basis of the updated flight times. If this update does not result in a conflict, then the planning remains as it is. Observe that since the column generation algorithm needs about 3 minutes to compute a planning for one day, this takes quite a lot of computation time. However, the computation strongly decreases when the replanning has to be performed later on the day. Since a larger part of the day is fixed then, the instance size becomes significantly smaller.

In most cases, the scheduling algorithms provide schedules with some unassigned trips. These trips are allowed to remain unassigned as long as they are not due. However, when a trip is still unassigned just before its starting time, the planners have to intervene and apply some specific measures. In the simulation we automated this intervention and we introduced an event 'acute unassigned trip' which handles the conflict by subsequently trying the following 'emergency measures':

Remove superfluous trip: At AAS it is a rule, that the departure of a flight is served by at least two bus trips, e.g. in case of two trips the first one leaves 25 minutes before departure and the second one 5 minutes before departure. If the number of passengers fits in one bus, the first of the two trips can be canceled if there is not enough bus capacity.

Change the break of a driver: If the trip can be performed by delaying the break of a driver, this is allowed as long as the driving time legislations are respected.

Extend the shift for a bus: There is a possibility to ask drivers to work overtime for 30 minutes, if this does not lead to more than 4.5 hours of work without a break.

Fix a trip to one bus: This measure assigns the unassigned trip to the bus that has been idle for the longest time at the moment this trip has to start. If this results in a conflict with a later trip, then the latter becomes unassigned.

If all the above measures fail, the trip remains unassigned, which is logged in the simulation.

To compare column generation and FCFS, we conducted experiments for five days for which flight, trip and shift data of AAS was available. These days form a subset of the days used for the experiments in the previous section. All five days represent different days of the week. Each day is simulated using both algorithms and for each algorithm the day is simulated 25 times. For column generation and FCFS we measured the robustness, the number of rescheduling steps, and the number and type of emergency measures. In the Tables 7 and 8 we report on the number of rescheduling steps and emergency measures respectively for each of the five days and their average.

Instance	Rescheduling steps			Ratio	
	CG	FCFS-CP	FCFS-NP	CG/FCFS-CP	CG/FCFS-NP
Day 5	107	120	85	0.89	1.26
Day 15	85	99	72	0.86	1.18
Day 20	13	142	99	0.80	1.14
Day 25	78	113	64	0.69	1.22
Day 30	58	83	49	0.70	1.18
avg ratio				0.79	1.20

Table 7: Mean and ratio for the amount of times the schedulers had to reschedule

Instance	Emergency measures			Ratio	
	CG	FCFS-CP	FCFS-NP	CG/FCFS-CP	CG/FCFS-NP
Day 5	17	30	23	0.57	0.74
Day 15	18	27	22	0.67	0.81
Day 20	14	41	31	0.34	0.45
Day 25	3	21	10	0.14	0.30
Day 30	4	13	8	0.31	0.50
avg ratio				0.41	0.56

Table 8: Mean and ratio of the sum of emergency measures used by the scheduling algorithms

The tables indicate that column generation outperforms FCFS-CP in terms of number of rescheduling steps and number of emergency measures. Although FCFS-NP requires fewer rescheduling steps than column generation, column generation strongly reduces the number of emergency measures that has to be used. This is probably due to the fact that column generation penalizes unassigned flights. When we look at the type of emergency measures that have to be used, we see that column generation uses fewer break changes and encouters fewer unsolvable unassigned trips.

6 Conclusion

We have presented an ILP representation for the bus planning problem at AAS and developed a new method for solving the problem. We have also written an implementation to test our algorithm and we were able to solve the schedule for any given day in a matter of minutes. Contrary to the current implementation at AAS, which makes use of a rolling

horizon of 3 hours, we are able to plan the complete day, which enables us to have a much better overview while planning for example the mandatory breaks. Furthermore, considering the full day enables us to investigate the combination of the bus planning problem with the gate planning problem, which we consider in (Diepen et al., 2009).

We looked at creating a robust schedule for a complete day of the bus planning problem under the assumption that the arrival time and departure time of all flights are static. To assess the robustness in more detail, we investigated the application of our algorithm for operational planning when changes in the arrival times and departure times occur. We performed a simulation study in which we compared the suggested approach with a simple first-come-first-serve heuristic (which forms also the basis of the program currently in use at AAS for solving the bus planning problem). Our simulation experiments indicated that compared to current practice the column generation algorithm reduces the number of rescheduling steps to 80 % and the number of specific conflict solving measures to about 40 %.

In the case of operational planning the schedule has be computed again every time a conflict introducing change in the data is presented by the Central Information System Schiphol (CISS). In operational planning there is less time available for solving the bus planning problem than in day-ahead planning. Therefore, adapting our algorithm to operational planning is an interesting topic for further research. One option is to use our algorithm in a rolling horizon approach, e.g, we compute a planning for the next 3 hours and repeat this at each conflict. Then we strongly speed up our algorithm but we lose the ability to benefit from the complete overview when planning the breaks.

As an alternative, when computing plans for a complete day, we do not want to have to solve the problem from scratch again every time a disturbance causes a conflict. We would like to reuse as much information from previous solutions as possible. One possibility is to keep all the bus plans in memory generated during the column generation process and when a flight gets a delay, update the trips corresponding to this flight in all bus plans to reflect the delay. After this we can remove all bus plans that became invalid due to trips overlapping with breaks or the end of a shift. In case a bus plan became invalid because of two trips overlapping after the delay of one of them, we could also create two new bus plans from this one invalid bus plan. Each of the two new bus plans would contain only one of the two trips that caused the original bus plan to become invalid. This way, the whole process of re-solving the problem would have a hot-start instead of starting all the way from scratch. The idea is that this would allow the problem to be solved in a much shorter time than when solved from scratch.

Finally, we may reduce the total time needed for solving the problem by making use of parallel programming. For different shift types, the pricing problems may be solved in parallel.

References

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2001). *Introduction to Algorithms, Second Edition*. The MIT Press/McGraw Hill.

Diepen, G. (2008). *Column Generation Algorithms for Machine Scheduling and Integrated Airport Planning*. PhD thesis, Department of Information and Computing Sciences, Utrecht University.

Diepen, G., van den Akker, J. M., and Hoogeveen, J. A. (2009). Integrated gate and bus assignment at amsterdam airport schiphol. In Ahuja, R. K., Möhring, R. H., and Zaroliagis, C. D., editors, *Robust and Online Large-Scale Optimization*, number 5868 in Lecture Notes in Computer Science, pages 338–353. Springer.

Diepen, G., Van den Akker, J. M., Hoogeveen, J. A., and Smeltink, J. W. (2007). Using column generation for gate planning at amsterdam airport schiphol. Technical Report UU-CS-2007-018, Institute of Information and Computing Sciences, Utrecht, the Netherlands.

Huisman, D., Freling, R., and Wagelmans, A. P. M. (2004). A robust solution approach to the dynamic vehicle scheduling problem. *Transportation Science*, 38(4):447–458.

ILOG (2005). ILOG CPLEX v9.1, `http://www.ilog.fr`.

Le-Anh, T. and de Koster, M. B. M. (2006). A review of design and control of automated guided vehicle systems. *European Journal of Operational Research*, 171(1):1–23.

Ropke, S. and Cordeau, J.-F. (2006). Branch-and-cut-and-price for the pickup and delivery problem with time windows. Technical Report C7PQMR PO2006-21-X, Centre for Research on Transportation (now: CIRRECT), Université de Montréal.