A heuristic approach to column generation for airline crew scheduling

Tomas Gustafsson

NO 1999-16/ISSN 0347-2809

1999
Mathematics
Department of Mathematics
Chalmers University of Technology
and Göteborg University
S-412 96 Göteborg, Sweden

Abstract

Airline crew scheduling gives rise to many difficult and interesting optimisation problems. With straightforward approaches to the crew pairing problem one specifically needs to solve very large set-covering problems. One suggested approach to these problems is column generation. In column generation, the choice of master solver can have a significant impact on performance, both with respect to speed and quality. We have investigated the possibility of using a heuristic integer optimiser for the master problem, and make a comparison with a simplex based algorithm. We have also tried to direct the generation process by using dual information from a network relaxation of the pairing problem. Finally, the combination of the integer optimiser and a simple subgradient algorithm gives a very competitive master solver which for most problems speeds up the complete solution process many times. Computational results are provided on several production instances from different airlines.
Key words : Column generation, Set-covering problems, Airline crew scheduling

Preface

This licentate thesis is the conclusion to the five semester ECMI post-graduate program (European Consortium for Mathematics in Industry). The program consists of number of core and specialised courses in the field of applied mathemathics, and an important part is the final project which should treat a specific industry application.

My application has been airline crew scheduling. The work has been performed within the ESPRIT-project PAROS (Parallel large scale automatic scheduling), which mainly aims at improving existing solution techniques through parallelisation and new algorithms. It has also to some extent been sponsored by the Swedish network of applied mathemathics (NTM), as part of their university-industry exchange program. Much of the work has been performed on location at Carmen Systems, who provides a commercial software used by most larger European airlines.

I would like to thank my two supervisors Dag Wedelin at Chalmers and Curt Hjorring at Carmen Systems, for help, support and many inspiring ideas. I also wish to thank Federico Hernandez, Olle Liljenzin and Tuomo Takkula as well as many others in the PAROS project, and at Carmen Systems. I would like to thank you all for your inspiration and time.

Contents

Chapter 1

Introduction

The airline industry has a long standing reputation for using several *Operations Research* (OR) techniques in its operations. Several of the planning problems in an airline lend themselves to mathematical optimisation. Due to the size and inherent complexity of these problems it is extremely difficult and time consuming, if not impossible to solve them manually. A driving force in the business is of course the enormous amounts of money that only a fraction of percent savings represents.

The application that will primarily be addressed here is airline crew scheduling, specifically crew pairing optimisation. These problems give rise to very large and difficult *integer programs*, primarily of the *set-covering* type.

The crew pairing problem is probably easiest described as the problem of planning crew schedules for a given set of flights. What is produced is a number of *pairings* which together cover all flights. A pairing consists of a number of *flight legs* or non-stop flights, which represent a possible and legal way for a crew member to work their way from home-base and back.

One of the major difficulties in airline crew scheduling is all the *rules* which have to be taken into account. There can be hundreds of different restrictions on how for example the weekly schedule of a pilot can be constructed. There are both government regulations and rules arising from corporate agreement.

A brute force approach to the *crew pairing* problem can be to enumerate all pairings, and then try to select the subset which have the lowest overall cost while covering all legs. Such an approach is of course only feasible for very small instances, since a fair sized problem easily have billions of possible pairings. The combinatorial explosion can be reduced using different heuristics, but this requires a deep understanding of the underlying structure to give good quality solutions.

There is on the other hand a significant advantage with a complete separation of the optimisation and the construction or *generation* of legal pairings, and that is in the modelling flexibility. With separate generation, any rule applying only to single pairings can be easily integrated and all solutions from the optimisation step will in this sense be legal.

Another suggested approach known as *column generation* tries to implicitly consider all possible pairings without actually generating more than a small subset. Here, the optimisation and generation are still separate algorithms, but they are much more integrated by using feedback from the optimisation in the generation step. Pairings are generated dynamically during optimisation, giving much smaller optimisation problems and potentially leading to significant speedup. The solutions are often of better quality and can for many problems be proven optimal.

A drawback with column generation is in the modelling flexibility. In the generation step a much more complicated subproblem is solved, and for this to work well some modelling limitations on the problem has to be imposed, especially with respect to the rules.

In column generation the main optimisation problems solved, known as the *master problems*, tend to be very degenerate and consequently difficult to solve. If not special care is taken to the choice of optimisation algorithms used, the method can exhibit poor performance both with respect to speed and quality. It is not only in the actual optimisation step this can occur, but the nature of the feedback sent to the generator strongly affects the rate of convergence.

The company Carmen Systems provides a software for airline crew scheduling which is used by most larger European carriers. In this system a new column generator is under implementation. We have here tried to use a heuristic integer optimiser to solve both the integer and linear relaxation version of the column generation master problem. In addition, an investigation into the impact of using feedback from the solution of an approximate pairing-model has been performed.

In chapter 2 a short introduction to the airline crew scheduling problem is given. The emphasis is of course on the pairing problem. We also describe the basic column generation approach used in chapter 3. A rather straightforward integration of the integer optimiser into this system is described in chapter 4 together with a brief overview of the optimiser itself. In chapter 5 we introduce a way of stabilising the generation step, based on an approximate network model for the complete pairing problem. A hybrid column generation master solver is presented in chapter 6, using the optimisation algorithm of chapter 4 and a simple subgradient approach. Final conclusions are given in chapter 7.

We give computational results for the different approaches on production problems from several different airlines.

Chapter 2

The crew pairing problem

There are quite a number of papers and books describing the OR problems arising in the airline industry, see for example [1]. The diversity in the type of problems is very large. There are difficult scheduling problems both for flying personnel and ground staff. Maximal utilisation of aircrafts is absolutely necessary to make the airline competitive. Allocation problems arise when trying to determine the location and size of different bases. The list can be made very long. One of the most studied problems is airline crew scheduling, and we focus on the part of this process known as the crew pairing problem.

To put things in perspective we will first in section 2.1 give a brief overall view of the airline planning process. The basic pairing problem is then described in section 2.2 and generalised in section 2.3, where we also discuss some possible solution approaches.

2.1 The airline planning process

It is probably the medium term planning step in an airline which is best suited for mathematical optimisation. With medium term we here mean a couple of months before day of operation. For the long term perspective, the decisions will mainly have to be of a strategic nature. In the short term the focus is on immediate problem solving where creating a solution which causes minimal disturbance to other operations is the highest priority.

With respect to medium-term planning one can identify four common steps, which are; construct the timetable, assign the fleet, construct the crew pairings and assign individuals to pairings.

- *Timetable*

  First a timetable is constructed. It is mainly commercial aspects which influence which flights to operate. The timetable must of course also be possible to operate with the current fleet. Therefore this step is not completely separated from the next, but far from integrated. Another aspect which becomes more and more important, as air traffic is steadily increasing, is the allocation of time slots at the airports.

- *Fleet Assignment*

  When the timetable is fixed the actual aircraft are assigned to the flights. The goal is of course to maximise utilisation. The aircrafts are divided into separate *fleets* depending on aircraft type. For each flight leg a certain aircraft type is usually demanded. An important side constraint has to do with aircraft maintenance and service. Within regular intervals the aircraft has to visit a maintenance depot for checkup and service.

- *Crew Pairing*

  With the timetable in place and aircrafts assigned to the flights, it is possible to consider the crews. The basic problem is of course, that each aircraft needs flight and cabin crew, for example a pilot, co-pilot and a number of flight attendants. The number of crew on a flight varies with aircraft type and sometimes with time of day. In this first step one does not consider named individuals. The crew pairings which are constructed and selected are typically one or several days long. They must of course be in compliance with rules and regulations, and together cover all the flights to be planned. Another important constraint is that there must be enough crew at each base with respect to the work-load.

- *Crew Assignment*

  After the crew pairings are constructed, the next step is to assign actual people to the pairings. This is a complicated process where most airlines use some sort of bidding system. It can be based on strict seniority or include an optimisation step where the objective is some sort of total "fairness", which can be rather difficult to model. In addition to actual flying time, there is also other work to be scheduled. There are always crews on standby in the case of illness, late changes or other disruptions to the schedule.

The planning process does not progress one step at a time in a linear fashion. There are constantly changes within all the four major steps, and plans are updated along the way. Changes can occur at any stage giving rise to what is known as different types of repair problems. If a flight is for example cancelled one must find a way to cover all flights without disturbing too many other crew pairings.

A really good solution to any of the four steps described above does not only have the lowest possible cost but is also robust. When there are changes, these should only affect a small part of the timetable, fleet or crew and not give rise to significant chain-effects.

2.2 The basic pairing problem

Typically the crew pairing problem is solved for each aircraft fleet separately, mainly due to different requirements on the crew. One also distinguishes between short-, medium- and long-haul problems. For short- and medium-haul it is the combinatorial explosion which is the main concern. The structure of the long-haul problems is quite different, and requires both another set of crew rules and agreements as well as a somewhat different solution approach.

In figure 2.1 we give a very small example of a pairing problem. The example only contains 6 flight-legs which are operated in exactly the same way every day. There are only 3 different airport, A,B and C. There are also very few rules to consider, and the objective is simply to minimise the number crew needed to operate all flights. (RTD means working-day in the figure.) For this problem there exist in total 18 possible pairings. The optimal solution consists of 2 pairings starting every day, with never more than 3 crews working simultaneously.

Ideally one would like to be able to formulate the pairing problem as a one-shot *integer program* (IP), where only the legs considered are sent as input. The optimiser would then decide which legs to connect, and output an optimised solution of complete pairings. Unfortunately, this is impractical due to the complicated rules and the often nonlinear cost function. Traditionally, pairing generation and optimisation are therefore done in complete separation. However, this leads to a combinatorial explosion in pairing generation and to very large optimisation problems.

Imagine that we could actually generate all possible legal pairings. The resulting optimisation problem can then in its simplest form be formulated as a *set-partitioning* problem:

$$\begin{gathered}\min cx \\ Ax = 1 \\ x \in \{0,1\}^n\end{gathered}$$

Each row of the binary matrix A represents a flight leg in need of a crew (the right hand side is 1). The columns are the pairings, with entries in the matrix only for the legs on which they operate. With each pairing a cost is associated, giving the cost vector c. A pairing/column is part of the solution if the corresponding variable x_i has the value 1 in the optimal solution.

If complete enumeration of pairings is used, A will have a huge number of columns (easily billions) and only a few hundred or thousand rows. (See for example [2] for an estimate of the number of pairings in some studied

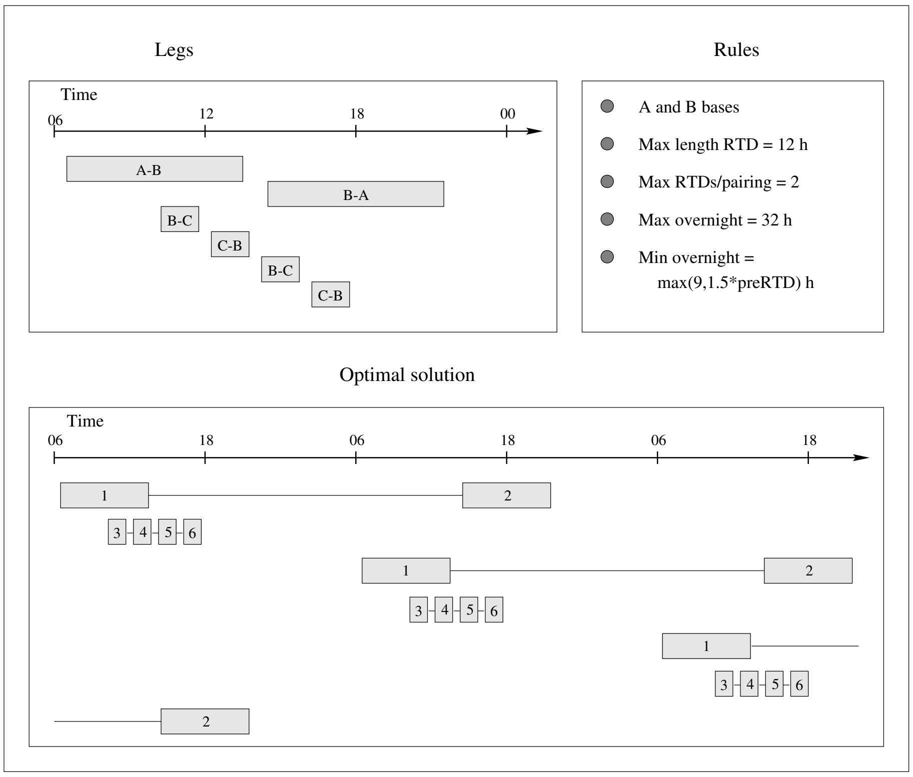

Figure 2.1: A small example

problems.) The resulting set-partitioning problem will be extremely difficult to solve, if not practically impossible. The generation process will also be very time consuming if at all feasible, especially since the many complicated rules and regulations must be taken into account.

2.3 Generalisation and the Carmen solution approach

When solving the real-life pairing problems, a proof of optimality is not the highest priority. To restrict the combinatorial explosion encountered when trying to generate pairings, one can apply different kinds of heuristics. This is mainly how the existing Carmen system works at this stage, and a more comprehensive description can be found in [3].

It is often possible to identify legs with only one or a few reasonable followers. If for example an aircraft lands at an airport with only one flight a day, it seems reasonable to follow the aircraft back again rather than overnighting. By locking such legs together one can reduce the problems significantly without losing much quality. A difficulty is how to determine good locks automatically. This can of course be left as a manual step to the planners, but it is time-consuming and the planner's experience will play an important role.

One can also restrict the problem by only considering a subset of the legs at a time, and then proceeding iteratively by successively including some more legs and removing some others. The legs can for example be selected using some time-window heuristic or be based on a number of aircraft. Local improvement schemes like this do of course not give any guarantee of global convergence, but have been shown to work well in practice, see for example [3].

Another possibility is to limit the number of legal follow-ons to a leg in the generation step. At a large airport there can be hundreds of potential legs to append to a pairing. By setting a limit on the number of follow-on legs to consider one can significantly reduce the number of columns generated. The difficulty with this approach is of course the choice of which leg-connections to consider. It is very difficult to tell in advance whether the problem is restricted too much.

Let us for the time being assume that we by using a number of intelligent heuristics, can limit the combinatorial growth in pairings enough to have manageable problems. The mathematical formulation above is still not enough to solve most real-life instances. There are two more modelling problems which must be addressed, namely *base constraints* and *deadheads*.

Most airlines operate from a number of *crew bases*, where in the normal case all crews must start and end their work-shifts or pairings. At these bases there are naturally restrictions on the number of available crew, which then give us the base constraints. They are most commonly formulated as limits on the total amount of work produced by the crews from a single base. The limit can be both upper and lower bounding.

A more difficult problem is the so called deadheading. It is for most airlines almost impossible to cover all flights without allowing passive legs on behalf of the crews. It is sometimes necessary, or often much cheaper, to have flying personnel travel along as passengers on single flights, rather than staying for a long time away from home base. It can even be worthwhile to use other carriers' flights. It is of course not practically possible to take into consideration as deadheads, all flights from all carriers. A certain subset must be selected without neither including or excluding too many. There has been some work in this area, see for example [4], but we will not here go into much more detail. We just note that we can not explicitly consider all possible deadheads, since the number duty periods and connections will then grow significantly.

A simple way to allow for more flexibility with respect to deadheads, is to change to a *set-covering* model rather than set-partitioning, that is allowing more than one crew on each flight. If a flight-leg is assigned to more than one crew, one is converted into an active leg and the rest to deadheads. This can of course cause legality problems, since all legs for different reasons are not allowed as deadheads. As an example, a deadhead can for certain rules not be legal in the middle of a *duty period* (or working-day).

Another reason for using a set-covering model has to do with performance. It seems like this model gives faster convergence for most larger problems. This can possibly be explained by the fact that when using a set-partitioning model and solving from scratch, it can be very difficult to find feasible solutions without adding some sort of dummy variables. The *linear programming* (LP) relaxation to the set-covering problem is also known generally to be more stable and consequently easier to solve.

The base constraints are simply added to our model, represented by the new constraints B. Note that these constraints can have entries in R^+ (often I^+), and be much denser compared to the set-covering constraints. The limit on the base production is represented by p, giving.

$$\min cx$$
$$Ax \geq 1, Bx \leq p$$
$$x \in \{0,1\}^n$$

In practice, it is often the case that the pairing problem is solved in an iterative manner with rising complexity. In the first step it is assumed that all flights are operated every day of the week in what is known as the *daily problem.* A very good solution to this simplified problem is constructed. The idea is to use this daily solution as input when solving the *weekly problem*, that is when one assumes all weeks are identical. The daily solution is said to be *rolled out* over the full week. In the last step the *fully dated* solution is constructed, where all irregularities in the planned flight schedule are taken into account. This way of progressively solving problems closer to reality, gives a benefit in that the produced plans have a high degree of regularity. However, there is by no means any guarantee that this approach gives optimal solutions.

Chapter 3

The Carmen column generator

Up until now we have discussed several different aspects of the pairing problem, but only one solution approach. So far we have suggested separate pairing generation restricted through heuristics and the solution of a set-covering problem. Another solution that has been proposed is column generation, where good pairings are constructed selectively with a constrained shortest path algorithm using dual input from the LP-relaxation of the set-covering problem. The pairing generation and optimisation are then in some sense still separate, but the improved integration of the two can lead to smaller optimisation problems and better solutions. However, the rules and the often non-additive cost function can be difficult to handle in the shortest path algorithm.

Column generation has proven to be an efficient method for solving crew pairing and other related problems. There is a wide selection of papers addressing the solution of the pairing problem on its own. A classic paper on the column generation approach is [5], which is probably the first published attempt on real-life instances. There has been many others following, for example [6] and [7].

We will first briefly discuss the basic idea in section 3.1. The column generator under implementation in the Carmen system is described in sections 3.2 and 3.3, with a short introduction to the specific problem we have addressed following in section 3.4.

3.1 Column generation basics

The word column refers to a variable and its corresponding column in the A-matrix in the optimisation problem, which for our application corresponds to a pairing.

The basic idea is to implicitly consider all possible columns/pairings, but only to explicitly generate a small subset of them. In the simplest case, a small set of pairings giving a feasible solution is taken as input. These pairings constitute the initial columns in the master problem. The master problem is formulated as a set-covering problem, just as in the previous section, but the solution vector x is allowed to take fractional values (i.e. the IP is relaxed to an LP). Feedback from the solution of this first master problem is then used as input to the generation subproblem known as the *pricing* step, where new columns are generated in some "intelligent" way. Only columns which can reduce the overall cost at this stage are accepted and added to the master problem, which is resolved, giving new input for the pricing routine and so on. The process continues until no more attractive columns can be found. The basic column generation idea is illustrated in figure 3.1.

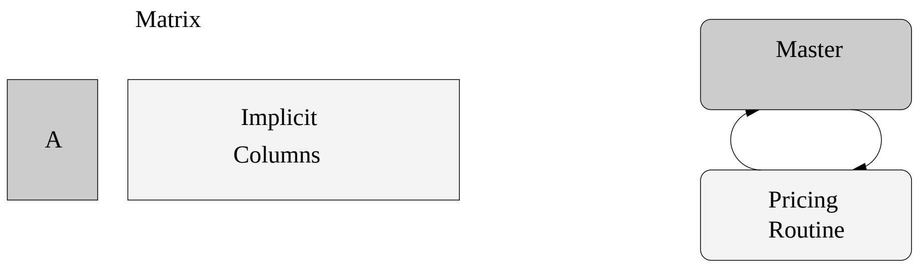

Figure 3.1: Basic idea behind column generation

Let us first take a somewhat closer look at the mathematics involved. The basic problem is formulated as in section 2.3, but we are starting with a very small set of columns and ignoring base constraints for now. We will present column generation from a *lagrangean relaxation* point of view, since this will be the chosen solution approach later on. The lagrangean relaxation of the set-covering constraints gives with the *multipliers* y the lagrangean function L:

$$L(y) = \min_{x \geq 0} cx + y(b - Ax)$$

It follows from LP-duality that the solution of $\max_{y \geq 0}(L(y))$, gives the same objective value as the LP-relaxation of the set-covering problem. Let A^0 be the initial matrix given, and y^0 the related optimal solution to the maximisation problem. If we now introduce a new column a_i with the cost c_i, for the minimisation problem we get:

$$\min_{x^0, x_i} y^0 b + (c^0 - y^0 A^0)x^0 + (c_i - y^0 a_i)x_i$$

If the *reduced cost* $(c_i - y^0 a_i) < 0$, we say that a_i *prices out.* For this set of lagrangean multipliers y^0, x_i will for the LP-relaxation obviously be part of the solution and consequently lower the total objective. The maximisation problem is then resolved with $A^1 = (A^0, a_i)$ giving new lagrangean multipliers or *duals*, y^1 which are used to price out new columns, and so on. If no more columns a_i with negative reduced cost can be found, we have a valid lower bound on the total objective and the process terminates.

We will from now on reserve the term column generation for the overall solution approach, and refer to the generation subproblem as the *pricing step.*

The generation subproblem which produces the new columns a_i is usually formulated as a *resource constrained shortest path problem* which returns the columns with the lowest reduced cost. To be able to get a reasonably general approach, special care has to be taken as to make very few assumptions on the cost functions and the many complicated rules.

Since most rules apply only to single working days or duty periods a natural approach is to use these as the building components of the network. That is, all duty periods are generated in advance, which is computationally expensive but still much cheaper than complete enumeration of the pairings. The duty periods can be seen as the nodes in the shortest path network and are only connected if it is legal to do so. the costs in the network are computed using the actual costs and the dual feedback y^k from the master problem, producing reduced costs.

What has been generated so far is a set of "cheap" columns which in an LP-sense constitute an optimal solution. To actually solve the crew pairing problem we do of course require integer solutions. A possibility is to try some branch-and-price scheme together with IP-heuristics which supply good upper bounds. For the most difficult instances one can not expect to solve the problems to proven optimality, but will have to be satisfied with the heuristic solutions.

3.2 The pricing step

The shortest path (SP) problem in the pricing step is solved on a duty based network where rules are modelled as combination of arcs and resources. An advantage with this setup is that everything generated will immediately be legal and the algorithms can be made reasonably efficient. A big disadvantage is in the lack of flexibility. First of all, it can be very difficult to model all rules correctly as resources, and even if possible it is not a trivial task to integrate them into a robust system.

Here, the choice has instead been to use a "black-box" legality system. Generated pairings are sent to a legality function which returns a yes/no answer and the pairings are either accepted or rejected. This complete separation of rules and generation does of course require some different modelling for the network and shortest-path algorithms. Since most of the more complicated rules and cost calculations apply to a single duty period, the pricing step in this system is also based on a duty network. In the simplest case, all duty periods are initially generated. This follows the ideas in most earlier work on column generation with application to to the pairing problem.

We will now in some more detail describe the Carmen pricing step and refer to [8] for more information. First, a simplified but impractical model will be described. We then discuss the modifications and extensions needed to solve production type instances.

There are two types of arcs and two types of nodes in the duty-network used to solve the pricing problem. Arcs either represent duty periods or legal connections between them (ground arcs). The nodes are either connection nodes or base nodes. A connection node has one incoming duty arc and a number outgoing connection arcs, or the other way around. The base nodes are the sinks, where pairings either start or end. (For this simple network, some arcs and nodes could easily be merged, but we choose to present the network in this way to simplify comparisons with the subsequent extensions.)

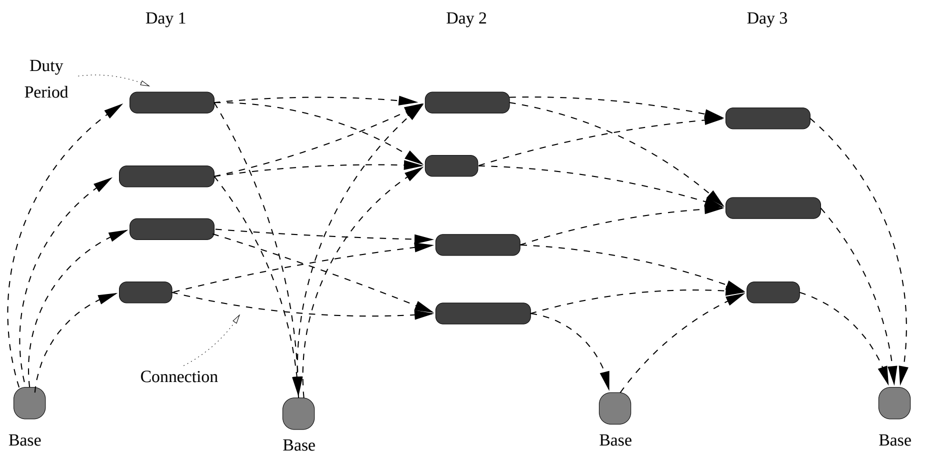

Figure 3.2: Full duty network

The number of duty periods can for some problems be quite large. In table 3.1 we give some statistics for a number of typical instances. The generation of all duty periods is possible except for some extreme problems (railways) and if not too many deadheads are fed as direct input to the generation process. The number of duty periods generated varies very much with the problem and the rule set.

Even though all duty periods are possible to generate, it is not feasible to explicitly consider all connections (ground arcs) between them. Even for a rather small problem the number of different duty period connections can be in the millions, which is also illustrated in table 3.1. In the table *el chunks* is the number of legs or sequences of legs locked together. For the problem *LH Paros 1* the generation of all legal duty period connections was not possible to do within a reasonable time-span.

Problem	el chunks	duty periods	dp conns
BA Cabin	290	5,235	275,808
BA FCS	250	10,648	1,512,903
LH FRA	268	5,438	254,382
LH Paros A320	195	3,790	352,229
LH Paros 1	927	144,624	???????
LH Paros 2 (hl)	927	24,269	20,000,000

Table 3.1: Duty period connections

The chosen approach to this difficulty is to instead use a relaxed network, where all rules applying to the duty period connections are not explicitly represented. As connection arcs we only consider connections between end- and start-legs of duty-periods instead of unique arcs between all connecting duty periods. This means that all duty periods with the same end-leg (start) use the same forward (backward) connection arcs. In figure 3.3 a small example is shown. Note that for a real-life problem, the number of duty periods which starts or ends with the same leg can easily be in the hundreds. The reduction in the number of arcs is significant and easily outweighs the addition of the nodes which are necessary to model the relaxed connections.

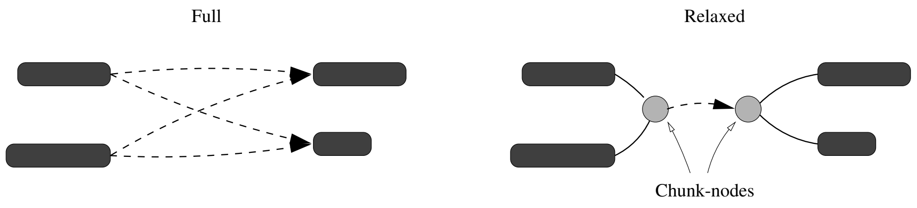

Figure 3.3: Full vs relaxed connection

The full network in figure 3.2 is relaxed and shown in figure 3.4. By this further relaxation of the pairing rules the risk of generating illegal pairings is of course increased, and for most problems this happens frequently. However, experimental results show that even for small problems where all duty period connections can be represented, the much more compact network will give a significant speedup.

With the network in place we now have the basics for solving the pricing problem. With each arc a cost is associated, both for the duty periods and connections (hotel costs etc). Using dual information provided by the master problem, reduced costs for the duty arcs are computed with respect to the chunks in that duty.

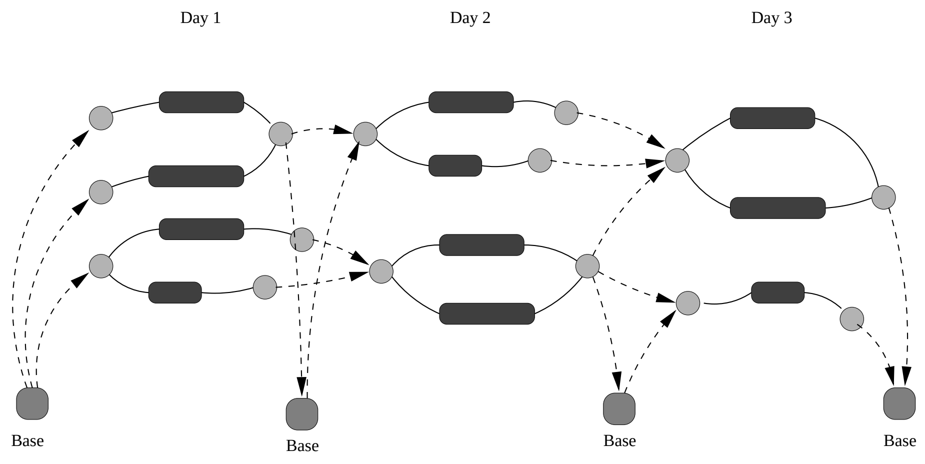

Figure 3.4: Relaxed duty network

On this network a k-shortest path problem is solved, following the approach of [9]. If the column/pairing generated is legal, it is directly added to the master problem. When this is is not the case, the second cheapest column is generated, checked and so on. The algorithm used to solve this k-shortest path problem is inspired by the work in [10].

If illegal pairings are generated, which is often the case, it is sometimes possible to take measures as to prevent the same rule violation from reoccuring. One can often identify *sub-chains* of pairings as the cause of the illegalities. If the rule which is not fulfilled spans at most two duty periods, the network is modified or *refined* to prevent this special combination again. This is done by splitting the connection nodes and introducing new and separate connection arcs.

At this stage only problems which have a strictly additive cost function, that is over the duty periods and duty period connections, are modelled correctly in the network. This assumption on the cost function is of course not always valid. It is not at all difficult to construct realistic cost functions which do not have this feature. The most common example from industry is when the crew's pay is based on a maximum over some sort of credit and the time spent away from home. Functions of this type are possible to model reasonably well, but we will for the time being stay with approximately additive cost functions.

3.3 Integration of pricing and master solver

At the end, what is required is of course integer solutions to the generated set-covering problem. In the Carmen system these solutions are primarily produced with the set-covering heuristic which will be described in the next chapter. In order to have better control over the generation process this heuristic is also called a few times during generation. A schematic picture of the complete Carmen column generator is finally shown in figure 3.5.

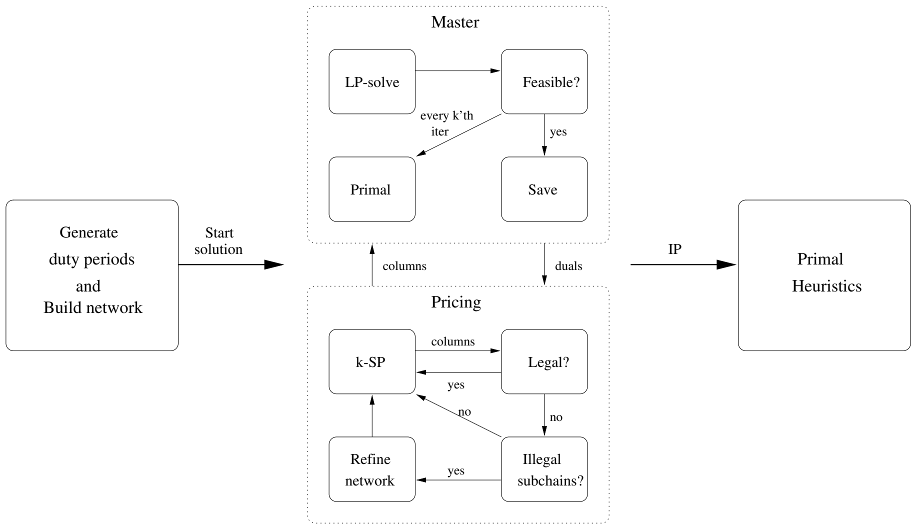

Figure 3.5: The Carmen column generator

As long as the cost function is approximately additive the column generator seems to work very well in practice. In many cases the first implementation of this column generator is beating the current more heuristic methods in the Carmen system, but it is not yet as flexible and doesn't have all the supporting functionality desired in a production environment. If the costs are not exactly additive, the lower bounds produced can of course not be completely trusted. The additive model can then both over- and underestimate the cost of a pairing. For the production rule-sets we have studied it seems like the costs are seldomly overestimated, but quite frequently somewhat underestimated. This may mean that many unnecessary columns are generated which can not lower the total objective, and subsequently performance suffers a bit.

3.4 Discussion

Although the Carmen column generator is already very competitive there is some room for further improvements which we have chosen to study more closely. There are sometimes difficulties with reaching the desired solution quality, or the solutions are not produced quickly enough.

As mentioned earlier the column generation master problems tend to be very degenerate, especially with respect to the sparsity and size of the problems. This is in some sense only natural if you consider the way they have been generated. Only the most attractive columns are added along the way, and except for the first few, all should be reasonably attractive also when solving the final IP.

Due to this degeneracy, resolving the master problem can be very time-consuming for some problems. The first tries at column generation did in most cases use a simplex-based master LP-solver. There has later been some work on using different LP-algorithms in a column generation setup. For this situation interior point methods seems to give some speedup compared to a simplex approach, see for example [2].

In the first implementation of the Carmen column generator a simplex-based master solver is used. This setup sometimes exhibits slow convergence primarily due to to the amount of time spent resolving the master problem. For some instances it also seems like a surprisingly large number of optimisation-generation loops is needed to achieve convergence.

There is of course also the problem with integer gaps. Since the column generation process as described here is completely based on LP-feedback, there is no guarantee whatsoever that the right columns for an optimal integer solution are generated. Some work has been done in order to reduce the IP-gaps, usually within the framework of branch and bound [7]. After convergence, the idea is to fix part of the problem and generate more columns selectively.

We have instead with the primary interest of generating higher quality solutions, integrated a set-covering heuristic as the master solver. This is described in the next chapter, with another modified version focusing on solution speed following in chapter 6.

Another important performance issue when using the relaxed SP-network described in figure 3.2, has to do with how much the network is refined. If the amount of refinement is large the pricing step will be seriously slowed down. The refinement in itself does of course take time, but it will also increase the memory requirements of the network all the way until convergence.

If the refinement is in a part of the network which is interesting for the final solution, the refinement is of course useful. Quite often the refinement occurs rather early on, and is of no use whatsoever for the final solution. Unfortunately this is not possible to decide in advance without a very good estimate of the optimal solution, and nothing is done to prevent it. We have in chapter 5 in some sense tried to find such an estimate and use it in a try to improve performance.

Chapter 4

A first integration of the integer optimiser

For the column generator master problem, it has been shown that interior point methods work very well both with respect to speed and quality [2]. This is primarily when compared to a straight forward simplex-approach. There has also been a number of different suggestions on how to deal with degeneracy in column generation in general, and with specifically the simplex-algorithm in mind. For example [11] report a considerable improvement by slightly modifying the master problem and stabilising the duals .

The heuristic integer optimiser mentioned in the previous chapter, has been used successfully for many years within the Carmen crew pairing system, to solve large set-covering problems. This system is used to solve the everyday production problems for several European airlines [3]. The heuristic produces many high quality *integer* solutions quickly [12]. In the Carmen system it is used as an IP-solver on its own, but has been shown to work well also as a primal heuristic in branch and bound-schemes [13].

Recent development has added some new functionality to the integer optimiser, and most importantly sped up the solution process many times. The new version of the integer optimiser, from here on called *PAQS*, combines the basic ideas from the original algorithm with an active set heuristic.

We have in the Carmen column generator used *PAQS* to solve the column generation master problem. The idea behind this is twofold. First, the algorithm is very fast and stable even for highly degenerate problems. Second, and most importantly, the dual feedback is generated in a rather special way, focusing more on integer solutions. This could reduce the IP-gap, without having to use some complicated branching strategy.

In sections 4.1 and 4.2 we describe the original integer optimiser and the most recent developments. The first integration with the column generator follows in section 4.3, and we briefly introduce the set of test problems we have chosen in section 4.4. Finally computational results and conclusions are in sections 4.5 and 4.6.

4.1 The integer optimiser

Since the basic problem to solve in this work is the set-covering problem we describe the algorithm specialised to this case. The basic integer heuristic in *PAQS* can be seen as a dual ascent algorithm with cost perturbations. It can also for certain parameter settings be interpreted as a dynamic programming or greedy approach [12].

The algorithm is based on lagrangean relaxation of the set-covering constraints. We have again the basic optimisation problem to solve:

$$\min cx$$

$$Ax \geq 1$$

$$x \in \{0,1\}^n$$

Where the entries in A are all in $0, 1$. The lagrangean relaxation is then:

$$L(y) = \min_{x \in \{0,1\}^n} cx + y(b - Ax) =$$

$$\min_{x \in \{0,1\}^n} (c - yA)x + yb =$$

$$\min_{x \in \{0,1\}^n} \bar{c}x + yb$$

$$y_j \geq 0$$

The components in $\bar{c} = c - yA$ are known as the reduced costs of the variables. For the sub-problem with fixed y the solution in x is:

$$x_i = \begin{cases} 1 & \text{if } \bar{c}_i < 0, \\ 0 & \text{if } \bar{c}_i > 0, \\ 0 \text{ or } 1 & \text{if } \bar{c}_i = 0. \end{cases}$$

In the *PAQS* algorithm a row-wise update of the reduced costs is performed. For every row with a given set of duals, we solve this subproblem so that only one of the reduced costs is negative, and perturb the costs in favour of this specific solution. The idea behind the perturbation is primarily to keep the reduced costs away from zero, giving a well defined primal solution. The one-row subproblem is:

$$\max_{y_j \geq 0} \min_{x \in \{0,1\}^n} cx + y(b - Ax)$$

$$A_j x \geq 1$$

Let r^- and r^+ be the smallest and second smallest reduced costs which have entries in the j'th row and with y_j set to zero. These values are projected on the positive half-line.

One primal and dual optimal solution to the one row subproblem is then given by,

$$y_j = \frac{r^+ + r^-}{2} \tag{4.1}$$

which gives a solution with one negative reduced cost. For the perturbation the following two different dual values are used,

$$y_j^- = y_j - \frac{\kappa}{1-\kappa}(r^+ - r^-) \text{ and } y_j^+ = y_j + \frac{\kappa}{1-\kappa}(r^+ - r^-) \tag{4.2}$$

where $\kappa \in 0, 1$ and y_j^- is used only for the variable with smallest reduced cost at this stage and y_j^+ is used for all the others. What happens is that the smallest reduced cost variable becomes negative if it is not already so, and is then shifted further away from zero in the negative direction. All the others are moved in the positive direction.

The algorithm performs the row-wise update over all rows iteratively. The same parameter κ is used during one pass over all the constraints, but can change between passes. There are several interpretations of the algorithm depending on the value of κ. For $\kappa = 0$ we have a plain row-wise dual ascent method. As κ approaches values close to 1 the algorithm will become more and more greedy. For $\kappa = 1/3$ the row-wise update is the same as would be the result of dynamic programming approach.

To get the best performance out of the algorithm the most critical strategy decisions has to do with the update of κ. By changing κ along the way, one could see the *PAQS* algorithm as a hybrid of the above mentioned classical approaches.

It is also possible to solve problems with more general constraints. For the pairing problem one typically has a number of base constraints, which restrict the workload for each base (see section 2.3). These correspond to more or less general linear constraints in the formulated IP/LP.

In *PAQS*, the general constraints are also updated one at a time together with the set-covering constraints. The update is similar to the one described above, but has some different characteristics for our specific application. The general constraints are usually a lot denser, that is they each affect a large number of columns. The single-constraint subproblem can then not always be solved to optimality, since it would be too time consuming. A heuristic is instead used to limit the sub-problem and to produce high quality solutions quickly.

4.2 The active set heuristic

Recent development has in the integer optimiser *PAQS* incorporated an *active set* heuristic, which has given significant speedup and is also possible to run in parallel on a network of workstations. A detailed description of the active set strategy and its considerable impact on the performance of the optimiser, is found in [14]. We here only give a brief summary and then discuss some aspects interesting with respect to column generation.

The heuristic is to some extent inspired by the ideas behind column generation and the general motivation behind it comes from the observation that in the optimisation algorithm, only a small set of variables has any effect on how the iteration proceeds. This is due to the fact that the constraint update is usually a function of at most two variables in the constraint. If the set of these critical variables was known in advance, it would in principle be possible to ignore all other variables and receive the same result much faster. This is obviously not possible, but it gives intuitive support for the idea of having a smaller number of variables that are likely to be critical selected in an active set.

The original algorithm then works only on the active set, and after a number of active set iterations, a *global scan* over all variables is done to see if more of them should become active. This update is based on a row-wise comparison of the reduced costs, where the selection criterion of the active set is based on whether a variable not yet in the active set would become critical for any constraint if this constraint was iterated next. This criterion was selected because it seems to be the closest one can come to the original algorithm, short of temporarily entering variables in the active set, which would not give the desired performance benefits. This has also been shown to work well in practice.

There are a few things worth emphasising with the active set update. First, a variable is activated if it has the smallest or second smallest reduced cost for all columns with entries in a single row. This means that variables with positive reduced cost can also be activated. Second, the optimiser asks for new columns dynamically while it is optimising before it has fully converged to an integer solution.

If one studies the behaviour of the active set update with respect to columns used in the feasible solutions, most interesting columns are brought in at an early stage, when there is no perturbation of the reduced costs ($\kappa = 0$). These columns could probably have been found also if we used some LP-algorithm for the active set. A few columns though, are only found attractive very close to convergence. That is, to get the best integer solution it seems likely that the introduction of the reduced cost perturbation is necessary. A representative example of when variables are activated is shown in table 4.2.

In the table we see the total number of active variables at different stages of the solution process, represented in the column *active total.* The number of variables added to the active set at different stages is illustrated in the column *active added.* In the last column *in integer sol* it is shown when columns in the final integer solution are brought into the active set. (Note that we are not presenting data in a linear fashion with respect to iteration number.)

These results have lead us to believe that *PAQS* could be used as the master solver in a column generation scheme, and then be able to generate a number of columns which are missed by a pure LP-based generator. The basic idea is of course to exchange the global scan over all columns in *PAQS* for the pricing step in column generation.

4.3 The first integrated master solver

We have chosen *not* to completely exchange the global scan in *PAQS* for the pricing routine. Instead we keep a buffer where new columns are entered, and old unattractive can be stored away. The active set is then updated and iterated in the exactly the same way as previously described, with a global scan over the buffer.

It seems worthwhile to do a number of active set updates and iterations before calling the pricing routine. This can be motivated by the fact that a few global scans over the buffer are often needed, before newly generated columns are brought into the active set and start affecting the duals. Figure 4.1 gives a schematic description of the *PAQS* column generator.

Since the PAQS algorithm seldomly solves the problem to dual optimality, we have no guarantee that very similar duals will not be produced several times. This means that the shortest path algorithm is likely to sometimes reproduce columns. The PAQS algorithm has difficulties with handling duplicate columns, and when PAQS is run stand-alone these are preprocessed away. In column generation mode, for every new column we dynamically check for duplicates, remove the old one if it is more expensive or do not add the new column if it has cost greater or equal to the old one. This also causes some changes in the pricing routine, which before relied on the duals being optimal.

The lower bounds from PAQS are usually fluctuating very much. This is not a problem when running the algorithm stand-alone as a set-covering heuristic, since the upper bounds usually develop in a stable fashion. When *PAQS* is used in the column generator, wildly fluctuating lower bounds and duals cause serious performance problems. The bottleneck is not directly in the optimiser. If very "strange" duals are sent to the pricing routine and the resulting columns are non-interesting, these are only put in the passive set. The difficulties are in the pricing routine. When using very extreme non-optimal duals, the pricing routine seems to generate many illegal rotations.

iterations	κ	active total	active added	in integer sol
0	0	630	629	50
10	0	1220	590	17
20	0	1744	524	19
30	0	2100	356	7
40	0	2261	161	7
50	0	2300	39	2
60	0	2303	3	0
70	0	2306	3	0
80	0	2308	2	0
100	0	2310	2	0
110	0	2312	2	0
180	0	2313	1	0
400	0	2314	1	0
470	0	2315	1	0
530	0.060000	2316	1	0
540	0.076667	2317	1	0
560	0.110000	2319	2	1
580	0.143333	2320	1	0
590	0.160000	2321	1	0
600	0.176667	2331	10	1
610	0.193333	2360	29	1
620	0.210000	2434	74	1
630	0.226667	2604	170	3
640	0.243333	2701	97	0
650	0.260000	2841	140	2
660	0.276667	3029	188	1
670	0.293333	3213	184	2
680	0.310000	3423	210	1
690	0.326667	3589	166	0
700	0.343333	3947	358	0
Integer solution found!				
710	0.356667	4085	138	0
730	0.373333	4139	54	0
740	0.385000	4173	34	0
750	0.401667	4217	45	0

Table 4.1: Variable activations for rail507.

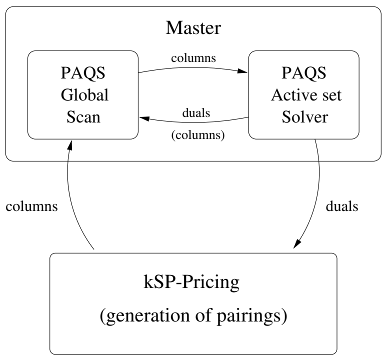

Figure 4.1: The first $PAQS$ column generator

This takes a lot of time, especially since the column generator is built on a relaxed network which is dynamically refined.

The behaviour of the lower bounds produced by $PAQS$ is for a typical column generation problem described in figure 4.2. For the duals produced, the lagrangean function actually takes on negative values in some iterations. The lower bound is of course always positive, and the lagrangean is cut at zero.

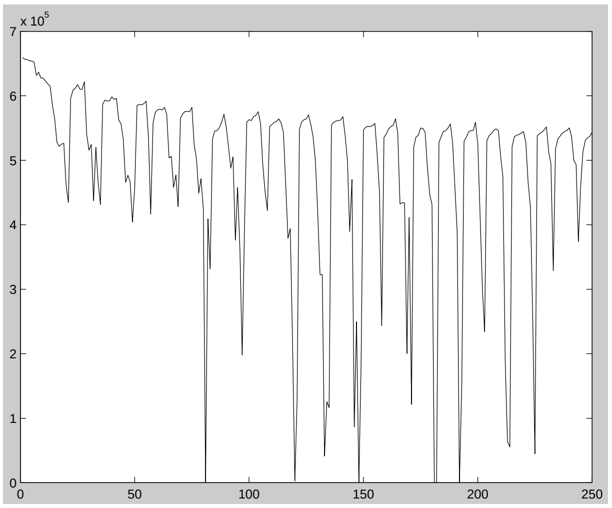

Figure 4.2: Lower bound fluctuations

To drastically reduce the difficulties with the pricing routine, we use the quality of the lower bound as filter for the duals. If the duals give a lower bound within a certain range of the upper bound, they are accepted and used for pricing. Otherwise more optimiser iterations are performed until acceptable duals are produced. Figure 4.3 gives a schematic description of this setup.

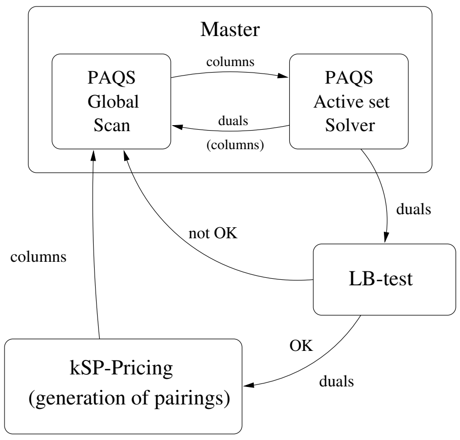

Figure 4.3: The *PAQS* column generator

A more difficult performance issue has to do with the termination criterion. When using a master solver which guarantees LP-optimality, convergence in the Carmen column generator is determined only based on the value of the reduced costs, i.e. the duals. If the LP/IP is very degenerate, the basic *PAQS* algorithm will have difficulties finding the optimal duals (or even get close to). That in turn will cause the generation process to go on long after most sensible columns are generated, and produce very large final IPs.

In order to somewhat tackle this problem, the termination criterion is based on a combination of heuristics. Termination is invoked if:

- No more columns with negative reduced cost are generated and a minimum number of iterations have been run.
- The objective is levelling out, that is if the lower bound is higher than for a long time and the primal solution has not recently improved.

4.4 Test problems

We have described how to integrate the *PAQS* algorithm as the master solver in a column generation setup as implemented by Carmen Systems. To properly evaluate this approach, and the ones in the following chapters 5 and 6, we require a representative subset of the problems solved in production. We have been careful to select test problems which represent different types of characteristics in rules, size, flight network, timetable etc.

In table 4.2 some statistics giving an idea about the size of the chosen problems are presented. The first data column *chunks*, is the number of single legs and legs locked together. The second column *rtds*, is the total number of different duty periods represented as duty-arcs in the network. Finally the column *dp leg-conns*, is the number of possible connections of duty-periods in the relaxed network represented by the connection-arcs.

The test sets have been taken from three European carriers. The first has a complicated rule structure and almost a "hub and spoke" network (five first problems). The second does in some sense represent the "normal" carrier with mathematically rather well-behaved problem instances and rule-sets, which are not so computationally expensive (problems *7567*, *all* and *prod*). The third carrier has rather uncomplicated rules, but the resulting optimisation problems are highly degenerate (problems *bm**). Both daily and weekly problems are addressed, indicated by a *dly* or *wly* in the problem names.

Problem	chunks	rtds	dp leg-conns
lh1 dly	881	73349	54124
lh9 dly	121	1215	822
a310 wly	1033	51066	45458
737 wly	2177	97844	54822
737 w6 wly	2177	49957	12081
7567 wly	1358	17316	2225
all wly	2052	53691	4745
prod wly	2548	69425	74361
bm dly	126	2762	2054
bm wly	778	43550	12493

Table 4.2: Test problems

All the test problems can be classified as short or medium-haul. The column generator successfully solves the long-haul problems as well, but the resulting integer programs are quite small and easily solved. For long-haul problems the dead-head selection strategies also play a more important role.

We have not yet incorporated base constraints directly into the column generation process when *PAQS* is the choice of master solver. In the test runs these are for now simply ignored.

There is so far no special dead-head selection strategy incorporated in the column generator. The only dead-heads considered are the ones given as input and the ones suggested by the overcovers in the final solution. Which dead-heads to select as input is not easily decided. In the Carmen system this is left to a heuristic known as the *matcher* which will not be described here.

All tests in this and the following chapters, have been run on a SUN

Ultra Enterprise 450 (4 * 296 MHz UltraSPARC 2) with 512 MB ram. The simplex solver used for comparison is CPLEX (version 6.01) and the chosen method is primal simplex with steepest-edge pricing.

4.5 Computational results

In this section we give an overview of the most important results and factors which affect the total solution time and quality or cost of the solutions. Besides the obvious comparison to the simplex based generator in total solution time and quality, a deeper analysis of the column generation process has been performed.

A very important thing to note is that the final IPs produced by the column generator are not in all cases solved to proven optimality. This is for many problems too time-consuming, and the heuristically produced integer solutions are usually of very high quality. This does of course make it a little bit difficult to analyse any differences in the solution quality when there are large integrality gaps.

In table 4.3 solution times and objective values for our test problems are presented. The first column *Total* presents the total CPU-time, including all setup-time, building the network, etc. The time spent in the pricing-routine is shown in the column *Gen* and the time in the master solver is presented in *Opt*. The column *Obj* represents the total objective value of the solution. A $(*)$ before the total CPU-time indicates that the heuristic termination criterion was invoked, i.e. the process stalled.

Execution times are for all non-trivial problems between two and eight times higher when *PAQS* is the optimiser choice. Note that for most problems, proportionally much less time is spent in the optimiser compared to pricing, when using *PAQS*. Termination is for half of the problems triggered by the heuristic criterion.

A typical behaviour for the bound development with *PAQS* is presented in figure 4.4 (problem *lh1 dly*). The lower bound fluctuates significantly and the upper bound is decreasing and finally levels out. The dashed line represents the cost of the best solution achieved when running with *CPLEX*. We see that this solution is reached long before termination.

Table 4.4 presents the lower bounds provided by the column generator for either choice of optimisers. The lower bounds with *PAQS* are all put in parenthesis, and the lower bounds used to compute the gaps are always the ones from *CPLEX*. For the problems which stalled, there are no valid lower bounds. For problem *all wly* there is a negative gap! This is due to cost overestimates in the pricing routine as the cost function is not completely additive. Unfortunately none of the test problems have strictly additive cost functions, which means that all the lower bounds will have to be considered as estimates. The solutions are significantly better and the approximate

Problem	Total	Gen	Opt	Obj
lh1 dly *PAQS*	24662	22191	741	709782
- " - *CPLEX*	10129	7775	802	712740
lh9 dly *PAQS*	286	248	6	186242
- " - *CPLEX*	209	179	1	186242
737 wly *PAQS*	*203517	194325	7703	3288049
- " - *CPLEX*	64228	57552	4582	3305330
737 w6 wly *PAQS*	36482	33716	2080	3335217
- " - *CPLEX*	19832	16517	2336	3351619
a310 wly *PAQS*	59427	57268	604	1272668
- " - *CPLEX*	7992	6191	254	1274728
7567 wly *PAQS*	1861	995	768	158772502
- " - *CPLEX*	723	477	156	158857582
all wly *PAQS*	*6611	2746	3627	212042374
- " - *CPLEX*	3057	934	1894	212070462
prod wly *PAQS*	*59695	53040	6255	67071221
- " - *CPLEX*	8195	5661	2126	67071568
bm dly *PAQS*	*415	328	84	571875
- " - *CPLEX*	107	94	10	559411
bm wly *PAQS*	*13387	6879	6445	3709392
- " - *CPLEX*	2327	320	1947	3752547

Table 4.3: Quality and CPU (sec)

integer gaps are for all instances except *bm dly*, much smaller when using *PAQS*.

To better understand why the *PAQS* setup is slower we have to take a closer look at what happens in the pricing step. In table 4.5 we give some statistics on the generation. We see that many more LPs are solved when *PAQS* is used. The most important difference with respect to performance is that many more shortest path problems are solved when *PAQS* is the choice for optimiser. This is also indicated by a higher degree of refinement of the network, represented by the column *Arcs (rel)* in the table. In this column we present the relative growth in the number of arcs in the network, comparing the number of arcs at convergence and in the initial network.

Note that each optimiser call to *PAQS* is not necessarily followed by a call to pricing. If the duals give a lower bound which is of too poor quality, another optimiser call is done. Surprisingly very few columns are regenerated in our test runs. This is probably due to the fact that the *PAQS* duals are fluctuating wildly.

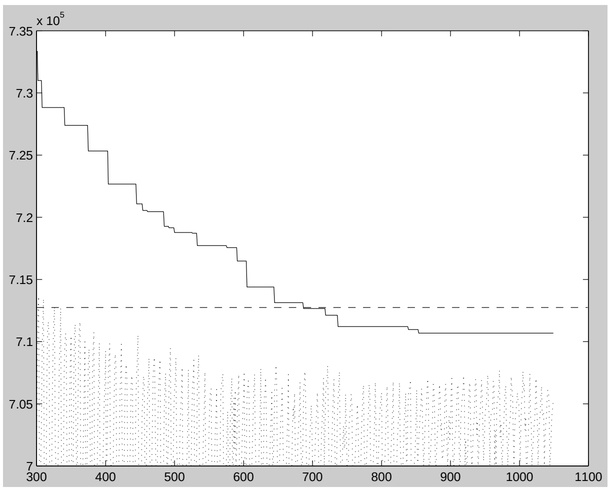

Figure 4.4: Upper and lower bound development

4.6 Conclusions

From the preceding computational results we can clearly conclude that the *PAQS* algorithm can be used to solve the column generation master problem. The solution quality is significantly better than what is achieved with a simple *CPLEX* implementation without any branching after LP-convergence. The lower bounds produced are on the other hand of a rather poor quality (several percents off).

The solution times required with *PAQS* are unacceptably high compared to the *CPLEX*-setup. If we take a closer look at where the extra time is spent, the conclusion is quite clear. Many more optimiser-pricing iterations are performed with *PAQS* as optimiser. The main reason for this is that when using *PAQS* to produce duals it is very difficult to use classical termination criterion. The duals from *PAQS* are quite far from optimal when the problem has grown enough. Different heuristic approaches have been tried to overcome this problem, but has only been partly successful. The *PAQS* setup is still several times slower than what is the case with *CPLEX*.

The *PAQS* algorithm is first of all a primal heuristic. A lot of primal solutions are generated “for free”, especially in the early stages of the generation process. If we look at the upper bounds and compare to the final solution produced when using *CPLEX*, we see that the *PAQS*-bounds often reach the same quality long before termination. If the runs were terminated at this stage the *PAQS*-setup would be more competitive with regard to speed, but still in most cases slower. It is of course also possible that the *CPLEX* runs could be terminated prematurely and still give the same integer solutions. This is somewhat difficult to say since very few integer

Problem	LB	UB	diff	gap (%)
lh1 dly *PAQS*	(704855)	709782	513	0.0723
- " - *CPLEX*	709269	712740	3471	0.4900
lh9 dly *PAQS*	(186133)	186242	9	0.00483
- " - *CPLEX*	186233	186242	9	0.00483
737 wly *PAQS*	(—)	3288049	14001	0.43
- " - *CPLEX*	3274048	3305330	31282	0.96
737 w6 wly *PAQS*	(3264691)	3335217	26111	0.79
- " - *CPLEX*	3309106	3351619	42513	1.28
a310 wly *PAQS*	1262473	1272668	1863	0.15
- " - *CPLEX*	1270805	1274728	3923	0.31
7567 wly *PAQS*	(149442147)	158772502	4230	0.00266
- " - *CPLEX*	158768272	158857582	89310	0.05625
all wly *PAQS*	(—)	212042374	-26596	-0.01254
- " - *CPLEX*	212068970	212070462	1492	0.00070
prod wly *PAQS*	(—)	67071221	3025	0.00451
- " - *CPLEX*	67068196	67071568	3372	0.00503
bm dly *PAQS*	(—)	571875	19035	3.44
- " - *CPLEX*	552840	559411	6571	1.19
bm wly *PAQS*	(—)	3709392	119566	3.22
- " - *CPLEX*	3589826	3752547	162721	4.39

Table 4.4: Lower bounds and approximate IP-gaps

solutions are produced dynamically when *CPLEX* is used. Naturally the primal heuristic could be called more often, but that will slow down the process significantly.

Since the SP-network is dynamically refined, it is very important that we quickly get close to the interesting part for the final solution. With *PAQS* the network is refined a lot more, which slows down the pricing step. The *PAQS* duals are also quite unstable which cause many uninteresting pairings to be generated. This is a problem we will address next in chapter 5, where we try to estimate the optimal duals and use them for the initial pricing calls.

Proportionally less CPU-time is spent in the optimiser when the choice is *PAQS* rather than *CPLEX*. The difference is quite significant and leads us to believe that if the number of calls to the pricing-routine could be reduced, we would still have a good chance of beating the existing system. Better methods are clearly needed to decide when to terminate the generation process. This will be the main topic of chapter 6.

Problem	Arcs (rel)	LPs	SP-calls	cols
lh1 dly *PAQS*	2.8762	1109	11041	17053
- " - *CPLEX*	2.0650	314	6562	8436
lh9 dly *PAQS*	1.3224	148	829	704
- " - *CPLEX*	1.1979	142	1164	765
737 wly *PAQS*	2.6765	2152	71007	101608
- " - *CPLEX*	1.9288	215	23871	20735
737 w6 wly *PAQS*	1.7548	964	35007	45897
- " - *CPLEX*	1.4508	151	15211	14006
a310 wly *PAQS*	3.3783	614	33684	19563
- " - *CPLEX*	2.6590	97	22425	8194
7567 wly *PAQS*	1.2045	773	4976	46803
- " - *CPLEX*	1.1279	237	2533	6627
all wly *PAQS*	1.2748	1770	10726	104455
- " - *CPLEX*	1.1354	473	4777	17109
prod wly *PAQS*	1.6418	3649	46563	118458
- " - *CPLEX*	1.1710	523	15332	24274
bm dly *PAQS*	24.4036	1292	2170	1537
- " - *CPLEX*	11.2044	466	1103	971
bm wly *PAQS*	5.1673	2899	46689	75437
- " - *CPLEX*	2.6668	379	14926	11877

Table 4.5: Generation statistics

Chapter 5

Stabilised duals for pricing

We concluded in the last chapter that the duals from *PAQS* are generally very unstable with respect to pricing. In this way columns that are useful in an IP-solution may be generated, but it causes performance problems in the pricing step. Because of the dual fluctuations many more than the really interesting columns are produced by the pricing routine. These will be useless in creating better bounds, both upper and lower. The dynamic refinement of the pricing network also makes it important to quickly get close to a reasonable solution. This is of course true with either choice of optimiser.

We suggest a stabilisation scheme for the duals used for pricing. Given an estimation of the set of optimal duals, we bias the duals produced by the master optimiser towards this solution before the pricing routine is called. To get an estimate of the optimal duals, we need to solve an approximation of the pairing problem. A requirement is of course that this approximation should be easy to solve, and still give reasonable results.

One obvious idea is to solve a highly restricted problem, where for example the number of connections to consider between legs are set to a minimum. One could even lock legs together, but it is then doubtful if the dual feedback is worth anything. The difficulty with restricting the problem is that it requires a deep understanding of the specific instance, to guarantee both feasibility and good quality.

Instead of further restricting the problem we suggest a relaxation. By relaxing some of the pairing rules it is possible to formulate the complete pairing problem as a *network flow* problem with side constraints. This follows the approach in [4], but our formulation is somewhat different, and is applied to another class of problems. They have with considerable success, primarily used their model to filter deadheads for long-haul problems. So far our primary interest is only in the initial duals which can be produced, and to some extent the lower bounds on the total cost which are also given.

The network flow model is described in section 5.1, with the dual stabilisation scheme following in section 5.2. Finally we give some computational results in section 5.3 and conclusions in section 5.4.

5.1 A relaxed network flow formulation of the pairing problem

In order to direct the generation process, we want to have an estimate of the optimal duals for the LP-relaxation of the crew pairing set-covering problem. In chapter 3 we noted that most complicated pairing rules apply only to single duty periods. If the rules which apply to chains of duty periods or complete pairings are ignored, the pairing problem can be formulated as a network flow problem with side constraints. This is a rather simple problem to solve and the LP-solution is for many of our test problems integer. Since what we solve is a relaxation, the solution objective will provide a lower bound on the complete pairing problem.

We use exactly the same network as the one used for the pricing routine and the restrictions are of course also the same. We are still using a black-box rule system where the only legality feedback available is a yes-no answer. This does again make it very difficult to use problem specific characteristics but allows for maximum flexibility. The cost function must as before be approximately additive over the duty periods for the solution to make sense, and the lower bound is only valid if it is strictly so, or if an additive model underestimates the true cost.

The network flow problem is formulated as an ordinary IP, where we can distinguish two types of constraints. First of all there are the set-covering constraints A which are represented by the side-constraints in the network flow model. Second, we have the actual flow constraints F. The coefficients in F_d are in $\{0,-1\}$ and F_c in $\{0,1\}$, and we call z the connection variables.

$$\min c_d x + c_c z$$
$$Ax \geq 1$$
$$F_d x + F_c z = 0$$
$$x_j, z_j \in \{0,1\}$$

The basic problem, putting crews on every flight, is modelled in the same way as before with one row for each flight leg, but with one column or variable x_i for each possible duty period. A coefficient in the matrix A is 1 if a leg is in the duty period, 0 otherwise. The right hand side is 1. Duty periods with no legal followers or predecessors are removed in a preprocessing step, and are never part of the actual network.

One variable z_i is created for each possible and legal night connection between two legs. The night connections at a base could for the basic model be excluded. However, variables for the base nodes give some added modelling flexibility and they simplify the construction of actual solutions.

Flow constraints are introduced to make sure that, in a solution, a duty period going in to a non-base airport will have another duty period going out. For each leg that arrives to, and departs from an airport, one row is added respectively. In these constraints, minus ones are put in the rows representing a duty period's end and start legs. For the night connection variables we put ones in the rows representing the legs of the connection (two entries per column). The right hand side is zero. For the single base problem we get a single commodity network flow problem with side constraints.

There are many ways to model the connection between duty periods. One could for example have a connection variable for each possible combination of duties. As described in section 3.2 there are millions of such possibilities even for moderate sized problems. If introduced here that would give the same number of connection variables. At the other end is a further relaxed connection model where we only consider connections at the same airport. Such a relaxation will of course give rise to many illegal connections.

We have chosen to use the same leg to leg connections as in the relaxed network which is used by the column generator. This is not only for the convenience of already having the network in place. For the same reasons as with the column generator network, these connections represent a reasonable compromise between the flexibility, efficiency and strength of the formulation.

Another reasonable compromise which is suggested by [4], uses time windows for the incoming and outgoing duties. The minimum connection time for a duty period is included already before continuations are considered. The drawback with using this model is that it requires specific knowledge about the rules and also that they are quite easily analysed.

For the network flow formulation, we have so far only presented a very simplified model of the crew pairing problem. First of all, for most problems there are more than a single base. The network model can quite easily be extended to handle several bases, but growth in size of the IPs because of this generalisation is not negligible.

To allow for multiple bases we use copies of all duty period columns, one copy for each base. Duty periods are not always legal for all bases, which means that the sub-matrices representing different bases will be slightly different. The same applies to the connection variables as well.

The flow constraints are separated according to base, which means that we introduce a large number of new constraints. These constraints will all form distinct groups representing different networks, one for each base.

With the network flow model just as with the column generator in general, we have some restrictions on the type of base constraints that can be handled correctly. Production constraints which are additive over the duty periods and connections are easy to handle. They are only applied to the duty period and connection variables, now distinct for each base. It is also possible to put a restriction on the number of rotations from a base, by using the base sink variables. The difficulty is with constraints on complete pairings. A base constraint that for example puts an upper limit on the number of rotations of a certain length in a solution, is not possible to incorporate.

If we allow for several bases and introduce the base constraints represented by B_d and B_c, we get the following problem:

$$\min c_d x + c_c z$$
$$Ax \geq 1$$
$$F_d^b x + F_c^b z = 0$$
$$B_d x + B_c z \leq p$$
$$b \in Bases, x_j, z_j \in \{0, 1\}$$

The matrix structure of a two-base problem is illustrated in figure 5.1.

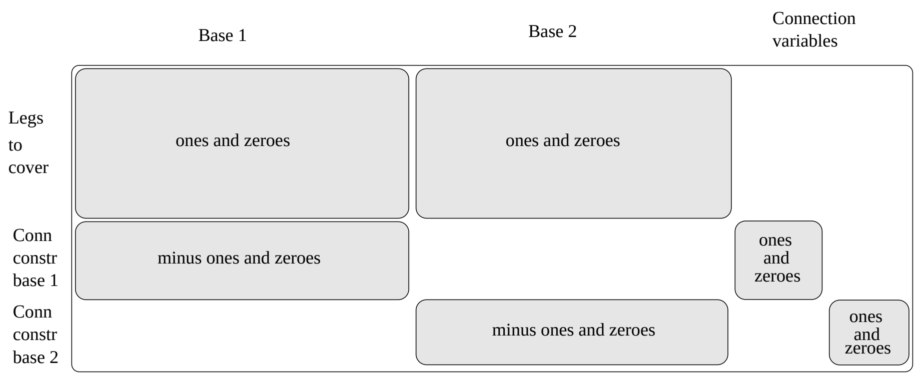

Figure 5.1: Matrix structure, multiple bases

Since the main idea behind using the network model is to speed up the complete column generation process, and not necessarily to create a very good lower bound, the actual solution time must be very low. For large instances, feeding the problem formulated as in the previous section straight into the LP/network-solver, creates problems with both size and degeneracy. This is especially the case when there are many bases, since we are using copies of all duty periods.

By allowing rotations to start and end at any base, the problem can again be formulated as a single commodity network flow problem. The set-covering constraints or side constraints, are of course still the same. All the base constraints will with this formulation unfortunately have to be ignored. It is not possible to distinguish which pairing belongs to which base. One could do some approximation, but it is very difficult to construct meaningful constraints while maintaining validity.

In order to ensure the validity of the lower bound the minimum cost of a connection or duty period must be computed over the bases, and then used as arc cost in the network. The costs are usually very similar so this is not a significant relaxation.

We also relax the integrality constraints since they are only meaningful with respect to the lower bound produced, and we are mainly interested in the dual feedback. The problem we have chosen to solve in order to get approximate duals to use in a pricing stabilisation scheme is finally:

$$\begin{gathered}\min c_d x + c_c z \\ Ax \geq 1 \\ F_d x + F_c z = 0 \\ x, z \geq 0\end{gathered}$$

The duals we will use for the stabilisation scheme in the next chapter is of course the duals corresponding to the set-covering constraints. One can quite easily see that these duals can be interpreted just as the ones from the ordinary version described in previous chapters. If the duals give a negative reduced cost to a "pairing" in the network flow version, it will give a negative reduced cost to this pairing in the shortest path network. (Provided of course, that the actual cost is additive)

5.2 The stabilisation scheme

The network flow relaxation of the pairing problem is solved once, before the actual column generation process begins. The master optimisation problem is solved in the same way as before. It is only for the pricing step that we directly manipulate the duals. When the reduced costs are computed for use in the SP-algorithm in the pricing routine, a weighted average of the duals from the master problem and the network flow relaxation is used.

We have chosen to let the weight of the network-duals decrease with the number of iterations, and finally be set to zero. Normally we set an upper limit on the number of iterations with the stabilised duals. By using non-optimal duals it is of course also possible to trigger premature termination, if not specific care is taken to the termination criterion.

Let y^{start} be the duals produced with network model and y^i be the current duals from the master problem given by either CPLEX or *PAQS* in the i'th iteration. The weight function $w(i)$ should satisfy $0 \leq w(i) \leq 1$ The duals used in the pricing routine $\bar{y}^i$ are then given by:

$$\bar{y}^i = w(i) y^{start} + (1 - w(i)) y^i$$

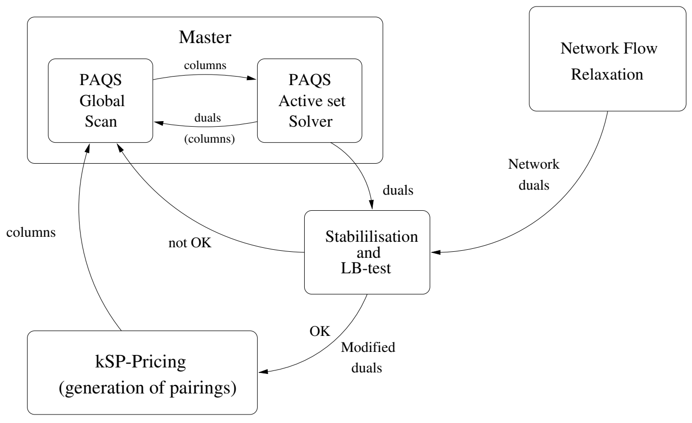

Figure 5.2: Stabilised column generation

(Since $y^i \geq 0$ (dual feasibility) we actually have that $\bar{y}^i \geq y^{start}$.)

One can imagine any number of different weight functions, but we have chosen a very simple one. Let C_1 and C_2 be constants of the same magnitude, not to large for the iterator i to have any effect, and I_{max} be the maximum number of iterations with any active stabilisation of the duals. The weight function is:

$$w(i) = \begin{cases} \frac{C_1}{C_2+i} & \text{if } i \leq I_{max} \\ 0 & \text{if } i > I_{max} \end{cases}$$

5.3 Computational results

The tests have been run on the same set of problems as in chapter 4. We have tried the stabilisation scheme using both *PAQS* and *CPLEX* as master optimisers. The network flow problem is always solved using the *CPLEX* hybrid network solver. When comparing the stabilised and the normal version of the pricing routine, we refer to the tables in section 4.5 for the results for the unstabilised version.

Comparing the results in table 5.1 and 4.3, we can see that the stabilisation gives some speedup for four of the problems when *PAQS* is used and for five with *CPLEX*. The rest are all slower. With *PAQS* four of the problems are terminated due to stalling, which makes it very difficult to compare these results (indicated by a $*$). The order of magnitude of the speedup is between 10% and 40 %. For the problems which are slower the differences are sometimes larger. For example the problem *all wly* solved with *PAQS* is three times slower. The relation in performance between *CPLEX* and *PAQS* does not change considerably with the introduction of the stabilisation.

The quality of the solutions is also somewhat different than what is the

Problem	Total	Gen	Opt	Obj
lh1 dly *PAQS*	16867	14627	406	710230
- " - *CPLEX*	16292	12734	874	713069
lh9 dly *PAQS*	281	242	8	186242
- " - *CPLEX*	228	194	2	186242
737 wly *PAQS*	*208652	199280	7442	3291661
- " - *CPLEX*	54943	49173	3273	3304459
737 w6 wly *PAQS*	37831	34473	2470	3338918
- " - *CPLEX*	13995	11328	1524	3349902
a310 wly *PAQS*	55648	53235	511	1272592
- " - *CPLEX*	7332	4639	191	1275050
7567 wly *PAQS*	3407	1399	1872	158769698
- " - *CPLEX*	710	440	144	158777694
all wly *PAQS*	*19995	7276	12251	212037150
- " - *CPLEX*	3230	732	2048	212069774
prod wly *PAQS*	*63883	52469	10616	67071194
- " - *CPLEX*	8144	5043	2380	67069423
bm dly *PAQS*	*548	410	133	574238
- " - *CPLEX*	130	114	11	558130
bm wly *PAQS*	*10968	4999	5808	3755323
- " - *CPLEX*	2901	1003	1748	3740569

Table 5.1: Solution time and total objective

case with the normal setup. In table 5.2 we again illustrate the quality by comparing the integer solutions with the lower bounds provided by *CPLEX*. Note again that the lower bounds are not always valid due to the non-additivity of the cost-function. This is obvious for the problems *7567 wly* and *all wly*, where the integer gaps when running with *PAQS* are negative! The solutions are better when using *PAQS* compared to *CPLEX* for all problems except for the two last, *bm dly* and *bm wly*.

We see that the solution cost is lower for four of the problems with *PAQS* and for seven with *CPLEX*, compared to the unstabilised version in table 4.4. This is somewhat correlated to the difference in speed, and it also works in reverse. For example *bm wly* with *PAQS* is solved much quicker with the stabilisation in place, but the solution is considerably worse.

In order to see if the development of the duals is more stable with the network flow start duals, we look at the dual difference from iteration to iteration. That is we computed $||y^k - y^{k+1}||_1$ and compare with the normal setup. This is for a typical example illustrated in figure 5.3. We see that the stabilised version is more smooth in the beginning. Around iteration 50 the weight of the network duals is set to zero, and the development is similar.

The growth in the pricing network is more or less on par with what

Problem	LB	Obj	diff	gap (%)
lh1 dly *PAQS*	(703423)	710230	834	0.12
- " - *CPLEX*	709396	713069	3673	0.52
lh9 dly *PAQS*	(184330)	186242	9	0.00483
- " - *CPLEX*	186233	186242	9	0.00483
737 wly *PAQS*	(—)	3291661	18125	0.55
- " - *CPLEX*	3273536	3304459	30923	0.94
737 w6 wly *PAQS*	(3273957)	3338918	30374	0.92
- " - *CPLEX*	3308544	3349902	41358	1.25
a310 wly *PAQS*	(1262032)	1272592	1786	0.14
- " - *CPLEX*	1270806	1275050	4244	0.33
7567 wly *PAQS*	(149669536)	158769698	-519	-0.0003
- " - *CPLEX*	158770217	158777694	7477	0.0047
all wly *PAQS*	(—)	212037150	-28466	-0.0134
- " - *CPLEX*	212065616	212069774	4158	0.0020
prod wly *PAQS*	(—)	67071194	2998	0.0045
- " - *CPLEX*	67068196	67069423	1227	0.0018
bm dly *PAQS*	(—)	574238	21398	3.87
- " - *CPLEX*	552840	558130	5290	0.96
bm wly *PAQS*	(—)	3755323	165497	4.61
- " - *CPLEX*	3589826	3740569	150743	4.20

Table 5.2: Lower bounds and approximate IP-gaps

to expect when looking at the solution times. This is illustrated in table 5.3. Worth to note is that the problem *all wly* solved with *PAQS*, does not exhibit a large growth in the number of network arcs but is still much faster without stabilisation. Instead many more LP-pricing loops are performed, meaning much more time spent in the master optimiser and pricing routine. Note also that better solutions with the stabilisation does not immediately imply that more columns have been generated.

Besides the start duals the solution of the network flow relaxation of the pairing problem also provides us with a lower bound. In table 5.4 we compare the network lower bound with the lower bound and integer solutions achieved through column generation with *CPLEX* and stabilised pricing duals. The lower bound from the network flow relaxation is for most problems approximately 10 % worse than the column generation lower bound. For the problems *7567 wly*, *all wly* and *prod wly* the bounds are only around 2% off. This can probably be explained by the fact there is only a single base for the first two, and just a few for problem *prod wly*. There are also many one day pairings for these problems. That is, the network flow relaxation should be a good approximation of the true problem.

Solution times for the network flow relaxation are rather low compared

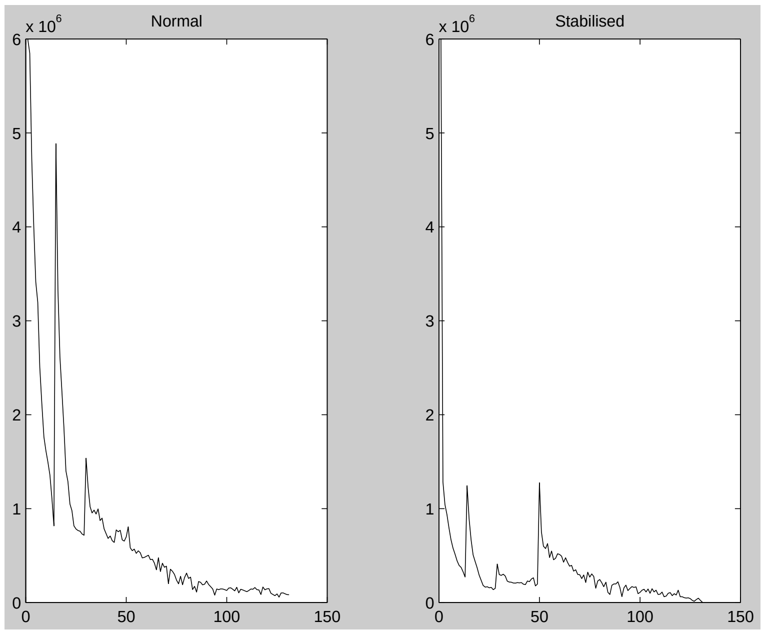

Figure 5.3: Dual fluctuations for problem *737 w6 wly*

to the total column generation time. The times also seem to scale rather well with the size of the problems.

5.4 Conclusions

The present stabilisation scheme proposed for the duals used by the pricing routine can not be concluded successful. For some problems the stabilisation gives a considerable speedup, for some it slows down the solution process. The solution quality is sometimes better and sometimes worse. One can maybe say that the stabilisation gives slightly better solution when run with *CPLEX*, but otherwise we have not been able to discover any pattern for when the stabilisation is likely to work. For the time being we must conclude the performance differences to be more or less random.

From the solution of the network flow relaxation of the pairing problem a lower bound is provided. This bound is for some problems of rather good quality but is very dependent on the specific problem structure.

The network flow formulation we have used can easily be strengthened and give better bounds. Solution times will then for most difficult problems be prohibitive with the solver we have used. We think it is worth investigating how a more efficient solution approach to this special network flow problem can be implemented. More of the underlying structure should be possible to use also in the basic formulation of the problem.

Problem	Arcs (rel)	LPs	SP-calls	Columns
lh1 dly *PAQS*	2.5837	885	9201	13523
- " - *CPLEX*	2.0417	335	6919	8718
lh9 dly *PAQS*	1.3117	181	1055	882
- " - *CPLEX*	1.1996	145	1042	808
737 wly *PAQS*	2.6071	2277	68878	101324
- " - *CPLEX*	1.7759	180	21018	18475
737 w6 wly *PAQS*	1.7769	1079	39746	50540
- " - *CPLEX*	1.3974	132	13013	13057
a310 wly *PAQS*	3.6791	537	31348	18902
- " - *CPLEX*	2.2409	94	16987	8437
7567 wly *PAQS*	1.2054	979	5989	60366
- " - *CPLEX*	1.1244	242	2574	6418
all wly *PAQS*	1.3007	2984	17068	178109
- " - *CPLEX*	1.1390	462	4777	17159
prod wly *PAQS*	1.6120	4040	49492	126936
- " - *CPLEX*	1.1771	512	15181	22969
bm dly *PAQS*	25.1594	1427	2277	1573
- " - *CPLEX*	12.4724	518	1189	1050
bm wly *PAQS*	5.0078	2203	40443	47489
- " - *CPLEX*	2.6383	379	14695	10055

Table 5.3: Generation statistics

Problem	CPU	LB (netw)	Gap (LP)	Gap (IP)
lh1 dly	187	650712	9.02	9.58
lh9 dly	1	172433	8.00	8.01
737 wly	497	2976194	9.99	11.03
737 w6 wly	207	2986181	10.80	12.18
a310 wly	164	1165031	9.08	9.44
7567 wly	35	154674492	2.65	2.65
all wly	212	208667841	1.63	1.63
prod wly	409	65793431	1.94	1.94
bm dly	2	486354	13.67	14.76
bm wly	90	3256497	10.24	14.86

Table 5.4: The network flow solution

Chapter 6

A hybrid master solver

We have seen in the two preceding chapters that it is difficult to use the integer optimiser *PAQS* as a master solver in a column generation scheme. *PAQS* seems suitable for solving the master problems in the early stages of the column generation process, and also helps in finding a few critical pairings which are missed when *CPLEX* is used. Unfortunately a very large number of uninteresting columns are also generated. The main problem was analysed in section 4.6 to be the difficulty in deciding a proper termination criterion. In chapter 5 we tried to avoid the worst of the dual fluctuations caused by *PAQS* during generation but this was not successful.

The integer solutions produced dynamically by *PAQS* do on the other hand reach very good quality early on. A natural idea seems to be a combination of *PAQS* with a more traditional optimiser approach. This would primarily be to give better grounds for a termination criterion, but also to help stabilising the whole generation process.

It has been noted by others, see for example [2], that subgradient algorithms perform well in a column generation context. We suggest to use a simple subgradient approach to produce the duals, when the pricing routine is not performing well using the *PAQS*-input. The decision on which optimiser to use is done dynamically, and often both versions are called at the same time.

The basic subgradient algorithm is described in section 6.1 and the integration with *PAQS* and the rest of the system in section 6.2. Computational results and conclusions follow in sections 6.3 and 6.4.

6.1 A simple subgradient approach to the set covering problem

Subgradient algorithms are a classical approach to many non-differentiable optimisation problems. For the set-covering problem there are a number of classical papers for example [15] and [16], and we refer to these for more detailed description. We will for this work stay with a very simple implementation which will be briefly described here.

Let us first restate the set-covering problem:

$$\min cx$$

$$Ax \geq 1$$

$$x \in \{0,1\}^n$$

We are now only interested in dual solutions, and will ignore whether the primal solution in x is feasible or not. Again we consider the lagrangean relaxation, where the lagrangean function is defined by:

$$L(y) = \min_{x \in \{0,1\}^n} cx + y(b - Ax) =$$

$$\min_{x \in \{0,1\}^n} (c - yA)x + yb = \min_{x \in \{0,1\}^n} \bar{c}x + yb$$

The minimisation problem in the definition of L is called the *Lagrangean subproblem*. Note again that this subproblem can be solved to optimality by setting $x_i = 1$, if $\bar{c_i} < 0$, and $x_i = 0$ otherwise. The dual problem can then be stated as:

$$\max_{y \geq 0} L(y)$$

It is easily seen that $g = b - Ax$ is a subgradient of L in y. The subgradient algorithm consists of a stepwise update of the duals in the subgradient direction:

$$y^{k+1} = \max(y^k + s^k g^k, 0)$$

In order to maintain dual feasibility, the duals are projected as to not contain any negative components. A well known theorem states that convergence is guaranteed if the step-length s^k satisfies the following condition,

$$0 \leq s^k \leq 2\frac{L(y^*) - L(y^k)}{||g||_2^2}$$

where y^* is the optimal dual solution. We do of course not know the optimal dual solution and approximate with the value given by the best known primal solution x^{k^*}.

$$s^k = \alpha^k \frac{cx^{k^*} - L(y^k)}{||g||_2^2}$$

Let C_1 and C_2 be constants in the range of approximately 100. We have chosen to define the value of α^k in the following way:

$$\alpha^k = \frac{C_1}{\sqrt{C_2 + k}}$$

The subgradient algorithm is terminated when $g = 0$ or the primal solution objective value coincides with the dual.

6.2 A combined *PAQS* and subgradient master solver

The main reason for combining *PAQS* and a subgradient algorithm is to give a better termination criterion. Given that we had a reliable quality test for the *PAQS* duals, the subgradient solver can be called when that test is not passed. The subgradient duals are optimal or very close to, and the traditional negative reduced cost termination criterion can in principal be used.

It is of course very difficult to construct a really good quality test, without knowing the optimal solution. One could instead always call the subgradient solver, and for the pricing step use the duals which give the best lower bound. This is somewhat inefficient since the *PAQS* duals often give good lower bounds when compared to the optimal, especially in the early stages of the generation process.

We have chosen a compromise where the subgradient algorithm is activated when the IP gap is very large or changing drastically. Using the IP-gap as a decision basis should be rather reliable due to the very good integer solutions dynamically produced by *PAQS*.

For the first subgradient iteration the duals from *PAQS* are used as input. It is then usually easy to drastically improve the quality of the duals in a few steps. For subsequent calls the duals from the previous iteration are used.

With this setup the subgradient solver is called quite often, which is very costly with regard to CPU-time if driven to optimality. To limit the effort spent in the subgradient algorithm, we put an upper limit on the number of subgradient steps done for each call. A more sofisticated and efficient subgradient algorithm could of course be implemented, but as the results in the next chapter shows, already this crude implementation gives a considerable speedup.

A special feature with using both *PAQS* and the subgradient solver at the same time, is that the upper bound needed by the subgradient algorithm is provided by *PAQS*. This makes the choice of step-length easier, and no new heuristic is needed for integer solutions in the subgradient solver.

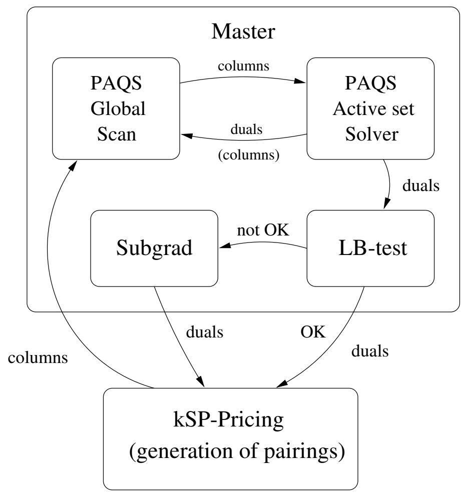

Figure 6.1: The *PAQS*-subgradient column generator

6.3 Computational results

Again we return to the set of test problems. In this section we only present the numerical results with the *PAQS*-subgradient optimiser as the master solver. For the other versions we refer to the previous sections 4.5 and 5.3. We have now reached the main computational result of this work.

The column generator with the *PAQS*-subgradient master solver is considerably faster than the *CPLEX* version for all problems except *prod wly*, *a310 wly* and the trivial *lh9 dly*. This is with respect to total runtime.

If we exclude setup time the speedup is even greater. The problems which require more computation time are still not much slower to solve. For the more degenerate problems with a simple rule structure (*bm dly* and *bm wly*), solution times are down to less than a third. Compared to the version with the *PAQS* optimiser on its own in chapter 4, the speedup can be more than ten times. For a more detailed comparison we refer to tables 6.3 and 4.3.

The solutions produced with the *PAQS*-subgradient setup are of the same quality as the ones produced with *CPLEX*. Looking at tables 6.2 and 4.4 we can see for problems with large IP-gaps that the *PAQS*-subgradient master solver is for most problems superior to *CPLEX* also with respect to quality. Compared to the results with *PAQS* as the only solver in chapter 4, the quality is somewhat worse.

The *PAQS*-subgradient column generator seems to give rise to the same type of pricing problems as when running with *CPLEX*, which is illustrated in table 6.3. There is of course a strong correlation between the number optimiser loops and columns generated and interaction with the rule system.

Problem	Total	Gen	Opt	Obj
lh1 dly	4110	2316	65	713261
lh9 dly	221	188	4	186242
737 wly	40830	38474	244	3307023
737 w6 wly	8752	7884	199	3346405
a310 wly	9244	7581	53	1274403
7567 wly	378	221	64	158774222
all wly	727	300	183	212077618
prod wly	11868	10945	519	67086650
bm dly	32	20	9	558111
bm wly	932	776	113	3721166

Table 6.1: Quality and CPU (sec)

Problem	LB	Obj	diff	gap (%)
lh1 dly	(709107)	713261	3992	0.56
lh9 dly	(186122)	186242	9	0.00483
737 wly	(318005)	3307023	32975	1.01
737 w6 wly	(3214424)	3346405	37299	1.13
a310 wly	(1269298)	1274403	3598	0.28
7567 wly	(158715734)	158774222	5950	0.00375
all wly	(211538970)	212077618	8648	0.00408
prod wly	(67013379)	67086650	18454	0.02752
bm dly	(551821)	558111	5271	0.95
bm wly	(3585164)	3721166	131340	3.66

Table 6.2: Lower bounds and approximate IP-gaps

The proportions are more or less the same for both optimisers. The generation of duplicate columns has unsurprisingly not been a problem. Very few duplicates are generated, using very little computation time (not illustrated in the table).

A question which naturally arises is whether the *PAQS* algorithm in its original form is at all necessary together with the column generator. Is it sufficient to use only the subgradient approach to solve the master problem?

This question is not easily answered since the use of the original algorithm provides the subgradient step with an excellent upper bound, which somewhat enhances performance. If not using the upper bounding functionality provided by *PAQS*, other possibly expensive heuristics must be developed. Throughout our tests *PAQS* does for most problems account for very little computation time, even for the very difficult instances. We have also noted that some solution columns are generated only when the *PAQS* feedback is used.

Problem	Arcs (rel)	LPs	SP-calls	Columns
lh1 dly	1.3671	139	2821	4127
lh9 dly	1.1885	105	889	740
737 wly	2.0681	193	25692	18587
737 w6 wly	1.5585	146	8752	15388
a310 wly	2.8366	90	21496	7264
7567 wly	1.1224	181	2105	6218
all wly	1.0931	288	3129	10738
prod wly	1.1533	473	14489	20135
bm dly	4.8556	231	534	747
bm wly	2.2295	248	10433	6467

Table 6.3: Generation statistics

As an experiment we significantly reduced the number of *PAQS* iterations and only used the subgradient duals in the pricing routine. This setup was more efficient than *CPLEX* on its own, but was outperformed by the *PAQS*-subgradient version both with respect to speed and quality.

We also note that the behaviour of the lower bounds produced by the basic *PAQS* and subgradient algorithm, are quite different. While the *PAQS* lower bound fluctuates, the subgradient lower bound behaves in a smooth fashion except for the very first iterations. It also appears that the *PAQS*-algorithm is well suited to solve the master problems in the initial stage of the generation process. The subgradient solver is on the other hand quite inefficient for these problems, but is very stable close to convergence.

The bound development of problem *7567 wly* is illustrated in figure 6.3. We have plotted the bounds achieved for each optimiser-pricing iteration. The dashed line is the upper bound, and the dotted is the lower, produced by *PAQS*. The solid line is the the subgradient lower bound. The subgradient optimiser is not always called which is why this bound is set to zero in the early iterations. We have chosen not to plot the first 50 iterations where *PAQS* very quickly solves also the dual problem.

In table 6.4 we summarise the results from running the *PAQS*-subgradient solver with the pricing stabilisation scheme from chapter 5. Here, as before, we can not distinguish any clear pattern. The stabilisation scheme speeds up the process in some cases but the solutions are then often slightly worse. There are two problems which stick out as behaving very different compared to the unstabilised version. The test problem *lh1 dly* requires more computation time with stabilisation, and the solution is significantly worse. For problem *prod wly* the situation is the reverse. It is solved in half the time with a better solution. Interesting to note is that this is the only problem where the normal *PAQS*-subgradient setup is considerably worse than the one with *CPLEX*.

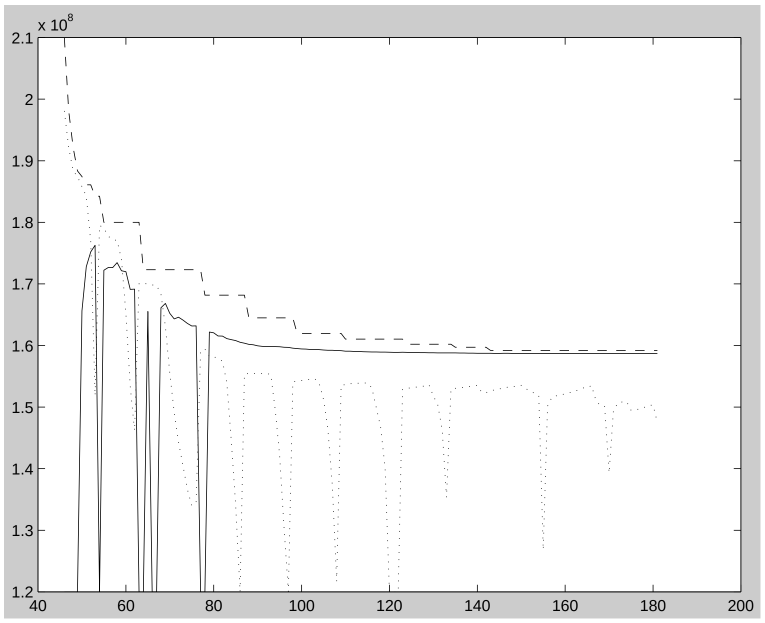

Figure 6.2: Upper and lower bound development for *7567 wly*

6.4 Conclusions

Column generation with the combination of *PAQS* and the subgradient algorithm as the master solver is on the average much faster and often give better quality solutions than a simple simplex-based approach with *CPLEX*.

The *PAQS*-subgradient setup is for most problems approximately twice as fast as the *CPLEX* version. If exclusively comparing the time spent in the optimisers, the difference is considerably larger. This lead us to conclude that our solver should be especially interesting when working with highly degenerate problems.

The *PAQS*-subgradient solver seems to be very stable. At the early stages of the generation process, *PAQS* very quickly either solves the problems to dual optimality or gets close to. At that stage the subgradient algorithm requires many iterations to reach the same quality. As the master problem grows in size, the *PAQS* duals are more often of poor quality while the subgradient duals are stable and of very good quality.

The solution quality is unfortunately somewhat worse than what is achieved when running *PAQS* on its own as in chapter 4, but the *PAQS*-subgradient setup is often ten times faster.

Unsurprisingly using the stabilisation of the pricing duals of chapter 5, with the *PAQS*-subgradient solver doesn't give any conclusive results.

Problem	Total	Gen	Opt	Obj
lh1 dly	5437	2725	37	715550
lh9 dly	240	193	2	186242
737 wly	35018	32301	177	3299285
737 w6 wly	8624	7368	105	3357645
a310 wly	9145	6617	41	1278360
7567 wly	463	267	65	158775270
all wly	894	278	166	212072930
prod wly	5133	4106	297	67082483
bm dly	41	27	9	55891
bm wly	818	566	95	3748127

Table 6.4: $PAQS$-subgradient solver with stabilised pricing duals

Chapter 7

Final conclusions

We have have investigated a number of heuristic approaches to column generation for the crew pairing problem. The focus has mainly been on how to solve the master problem, but we have also tried to stabilise the input to the pricing step directly.

In chapter 4 we used the integer heuristic *PAQS* on its own to solve the master problem. For this first integration with the column generator the primary objective was to improve the solution quality, and which have been successful. Unfortunately this implementation has serious performance problems with regard to speed. Many problems are solved 10 times faster when the master solver is *CPLEX*, and the cost reductions are not large enough to motivate such a difference.

When the column generator is based on a relaxed duty network, it becomes more important to quickly get close to reasonable solutions. This is due to the dynamic refinement of the network which has a strong impact on performance. In chapter 5 we tried to direct the generation process by using dual feedback from a network flow relaxation of the pairing problem. The idea was to avoid much of the refinement by having a reasonable estimate of the final solution already from the start. The results are mixed. The stabilisation scheme we suggest in chapter 5 does sometimes improve performance and sometimes make it worse. This is true for either choice of master optimisers.

From the network flow relaxation in chapter 5 a lower bound on the total cost is also provided. The bound is for most instances quite far off from the optimal, but can for some problems be a very good approximation. This is very much dependent on the flight and rule structure of the specific fleet in mind.

Finally in chapter 6 we suggest a combination of *PAQS* and a simple subgradient algorithm as an approach to the column generation master problem. This hybrid master solver is very competitive. Compared to the setup with *CPLEX* the new solver speeds up the complete solution process for most problems, and is often several times faster. The *PAQS* and subgradient algorithms seem in some sense to be complementary with respect to solving the dual problem. Some of the quality improvements in chapter 4 are unfortunately lost, but costs are still on a par with the ones produced with *CPLEX*.

Finally we would like to emphasise what we see as the main contributions of this work. First, with the introduction of the combined master solver with *PAQS* and the subgradient algorithm, we have improved the performance of the Carmen column generator significantly. This setup is also very stable and produces many integer feasible solutions dynamically. Second, we have in the column generator investigated a new approach for improving the solution quality, without using any direct branching.

Another important point common for all tests, is that they have been run on production instances with no rule simplifications or other restrictions. This is also true for the network flow relaxation in chapter 5, where as for all tests the only limitation is the duty additivity of the cost function.

For the future we first of all intend to incorporate base constraints in the *PAQS*- subgradient optimiser, which ought to be rather straightforward. What is interesting here is that the base constraints tend to change the solution characteristics and this might have an impact on performance in either direction.

More interesting is to study some possibilities for branching techniques that can be based on feedback from the *PAQS* algorithm. There are also a few apparent ways in which to strengthen the network flow relaxation of chapter 5 which should give better bounds and dual feedback. With this in mind, a specialised solution approach for this formulation can be fruitful to investigate.

Bibliography

[1] G. Yu, ed., *OR in the Airline Industry.* Boston, London, Dordrecht: Kluwer Academic Publishers, 1998.

[2] R. Anbil, J. J. Forrest, and W. R. Pulleyblank, "Column generation and the airline crew pairing problem," in *Documenta Mathematica – Journal der Deutschen Mathematiker-Vereinigung*, no. III, (Berlin), pp. 677–686, 1998.

[3] E. Andersson, E. Housos, N. Kohl, and D. Wedelin, *OR in the Airline Industry*, ch. Crew Pairing Optimization. Boston, London, Dordrecht: Kluwer Academic Publishers, 1998.

[4] C. Barnhart and R. G. Shenoi, "An alternate model and solution approach for the long-haul crew pairing problem," Jul 1996.

[5] S. Lavoie, M. Minoux, and E. Odier, "A new approach for crew pairing problems by column generation with an application to air transportation," *European Journal of Operational Research*, vol. 35, pp. 45–58, 1988.

[6] J. Desrosiers, Y. Dumas, M. Solomon, and F. Soumis, *Handbooks in Operations Research and Management Science*, ch. Time Constrained Routing and Scheduling. North-Holland, 1995.

[7] C. Barnhart, E. J. Johnson, G. L. Nemhauser, M. W. Savelsbergh, and P. H. Vance, "Branch-and-price: Column generation for solving huge integer programs." Working paper, January 1996.

[8] PAROS consortium (Carmen), *D3.3 Integration of internal modules*, Dec 1997.

[9] R. Marsten, "RALPH: Crew Planning at Delta Air Lines," *Technical Report. Cutting Edge Optimization*, 1997. to appear in *Interfaces.*

[10] E. Martins and J. Santos, "A new shortest paths ranking algorithm," *Tecnichal report: Universidade de Coimbra, Portugal*, 1995.

[11] O. du Merle, D. Villeneuve, J. Desrosiers, and P. Hansen, "Stabilized column generation," *Les Cahiers du GERAD*, 1998.

[12] D. Wedelin, "An algorithm for large scale 0-1 integer programming with application to airline crew scheduling," *Annals of Operations Research*, vol. 57, pp. 283–301, 1995.

[13] A. Atamtrk, G. Nemhauser, and M. Savelsbergh, "A combined Lagrangian, linear programming and implication heuristic for large-scale set covering partitioning problems," 1995. Working paper.

[14] P. Alefragis, P. Sanders, T. Takkula, and D. Wdelin, "Parallel integer optimization for crew scheduling," *Submitted to Annals of Operations Research*, 1999.

[15] M. Held, P. Wolf, and H. P. Crowder, "Validation of subgradient optimization," *Mathematical Programming*, vol. 6, pp. 62–88, 1974.

[16] M. L. Fisher, "The Lagrangian relaxation method for solving integer programming problems," *Management Science*, vol. 27, no. 1, pp. 1–18, 1981.