Annals of Operations Research 127, 203–222, 2004

A Decision Support System for Crew Planning in Passenger Transportation Using a Flexible Branch-and-Price Algorithm

RICHARD FRELING and RAMON M. LENTINK * rlentink@fbk.eur.nl
Erasmus Center for Optimization in Public Transport (ECOPT), Erasmus University, Rotterdam, The Netherlands, and
ORTEC bv, Gouda, The Netherlands

ALBERT P.M. WAGELMANS
Erasmus Center for Optimization in Public Transport (ECOPT), Erasmus University, Rotterdam, The Netherlands

In memoriam: It is with deep sadness that we announce that Richard Freling has passed away.

Abstract. This paper discusses a decision support system for airline and railway crew planning. The system is a state-of-the-art branch-and-price solver that is used for crew scheduling and crew rostering. Since it is far from trivial to build such a system from the information provided in the existing literature, technical issues about the system and its implementation are covered in more detail. We also discuss several applications. In particular, we focus on a specific aircrew rostering application. The computational results contain an interesting comparison of results obtained with the approach in which crew scheduling is carried out before crew rostering, and an approach in which these two planning problems are solved in an integrated manner.

Keywords: decision support systems, crew planning, branch and bound

1. Introduction

Crew planning for passenger transportation has received a lot of attention in the Operations Research literature, and only recently, cases are reported where companies in the bus, railway and airline industry are using advanced OR techniques for solving crew planning problems (almost) optimally. With the rapidly increasing computer power in the past decade, advanced OR techniques such as column generation are gradually becoming more and more applicable to real life crew planning problems (see, e.g., (Day and Ryan, 1997; Andersson et al., 1998; Desrosiers et al., 2000; Ryan, 2000)). Crew planning occurs on several levels, depending on the length of the planning period and whether the planning is for strategic, tactical or operational pur-

* Corresponding author. Room F1-11, Burg. Oudlaan 50, P.O. Box 1738, 3000 DR Rotterdam, The Netherlands.

poses. The two most widely applied crew planning problems are the crew scheduling problem for grouping tasks into duties, and the crew rostering problem for assigning duties to weekly, monthly or seasonal rosters for individual crew. Other crew planning problems are seniority based. Here senior crew dominates junior crew during the construction of the rosters. These problems include the bidline problem and the preferential bidding problem; see, e.g., (Gamache et al., 1998). In this paper, we will focus on crew scheduling and crew rostering problems.

1.1. Definition

The crew scheduling problem (CSP) is formally defined as follows: given a set of *tasks* (trips, flights, etc.) with fixed starting and ending times and locations, and given a set of *rules* and *criteria*, find the *minimum cost* set of *duties* such that each task is included in a duty and all rules are satisfied. A task is the smallest amount of work that can be assigned to one crew. Each duty consists of a sequence of tasks that satisfy certain rules like maximum work time, minimum rest time, etc. The cost function consists of a weighted sum of several criteria, such as the number of duties or the total deviation of average work time. An example of a task is Amsterdam 9:10 – London 10:50, while an example of a duty is Amsterdam 8:30 – Amsterdam 17:00. A duty may also cover more than one day, which is then often called a *pairing* in the aviation industry.

The crew rostering problem (CRP) is formally defined as follows: given the same information as for the CSP, and given a set of crew with certain *characteristics*, find the *minimum cost* set of *rosters* such that each task is included in a roster, all rules are satisfied, and the characteristics of each crew are taken into account. The cost function for the CRP contains elements from crew welfare, such as balancing workload, and from efficiency, such as minimizing layovers and crew deadheading. Just like a duty for the CSP a roster is a sequence of tasks that satisfy certain rules. The difference is that individual crew characteristics are taken into account for the CRP and the CRP typically spans one or two weeks. Examples of such characteristics are qualifications, pre-assigned tasks, individual requests, and the past rosters for the crew. The crew's past is necessary for checking laws and optimizing criteria that extend beyond the planning period. Usually, the CSP is solved first and the resulting duties serve as tasks for the CRP. We have experimented with integrating these two problems, where rosters are constructed directly from the tasks that served as input for the CSP in the definitions above (see section 5).

1.2. Contribution

Crew scheduling and rostering problems are usually modeled with set partitioning type of formulations, and solved with column generation techniques. The focus of this paper is not on these basic models and techniques for crew scheduling and rostering. Many papers have already been published on column generation techniques for crew planning, see, e.g., (Desaulniers et al., 1997; Vance et al., 1997) for recent papers on crew scheduling, and (Gamache et al., 1999) for a recent paper on crew rostering. However, we found that it is far from trivial to build a decision support system for crew planning from

the information provided in the existing literature. Although practical applications are considered in the theoretical papers mentioned above, most practical details (often also complicating ones) are left out, or at least it is not mentioned how these are incorporated in the computer system. Anyone who has implemented a branch-and-price algorithm will confirm that it is a tedious task to do so, taking a lot of effort, and trial-and-error. It requires a combination of experience with designing algorithms and implementing decision support systems. Therefore, the focus in this paper is on how to implement the aforementioned models and techniques successfully in a decision support system, taking all kinds of practical issues into account. To be more precise, the main contribution of our paper on a practical level is twofold:

1. We show that complicating details arising in practice can often be incorporated in a branch-and-price algorithm because column generation is a very flexible technique. Sometimes this introduces a heuristic feature in the algorithm, but this is generally not a problem in a practical context.
2. We provide insight into implementation issues, and therefore make it easier for novices in this field to implement branch-and-price algorithms.

Other papers describing how to implement details from practice are (Barnhart et al., 1998; Desaulniers et al., 1998a, b). Furthermore, additional implementation issues can be found in (Desaulniers, Desrosiers and Solomon, 2002; Barnhart et al., 1998).

Another contribution on a more theoretical level is that we perform experiments with solving rostering problems directly from tasks for an airline application, without first solving scheduling problems, thus integrating crew scheduling and crew rostering. Caprara, Monaci and Toth (2001) deal with a similar integration for railway applications, but they use a different algorithmic approach.

We base our experience on the development at ORTEC bv, the Netherlands, of a Decision Support System (DSS) for crew planning called Harmony CDR (Crew Duty Rostering). The system was originally designed to create and maintain rosters for airline cockpit and cabin crew, and later also for crew working on trains. In 1998 a new abstract implementation of the automatic planning tool in the DSS was started. The purpose was to set up a tool that can easily be used for different clients and applications, and that contains state-of-the-art mathematical techniques. Many practical issues came up during the development of the system for different applications. In section 4, we discuss several implementation issues.

1.3. Outline

In the next section, we discuss the mathematical background of the decision support system for crew planning. Several general algorithmic aspects will not be discussed here because they are already presented well in the aforementioned papers. Therefore, we briefly describe the underlying mathematical model and a branch-and-price algorithm. In section 3, we discuss the functionality and purpose of the system, and the classification of constraints and objectives used in the system. In section 4, we focus on the

implementation issues. In particular, we consider abstract implementation issues, the efficient use of computer memory and the reduction of run time. Four applications are discussed in section 5, of which one airline crew rostering problem in more detail. Finally, we summarize our work in section 6.

2. Mathematical background

In this section, we briefly consider the underlying mathematical formulation and the solution methodology used in the system. Throughout this paper we assume that the reader is familiar with column generation and branch-and-price. See, for example, (Barnhart et al., 1998) for a general discussion.

2.1. Set partitioning formulation

The set partitioning model (SP) presented below can be used to formulate most crew planning problems arising in practice. Let the set R denote the set of all feasible duties or rosters, let I denote the set of tasks which need to be covered. SP contains binary decision variables x_r, which equal 1 if duty or roster r is selected and 0 otherwise, and continuous decision variables s_i, which equal the number of uncovered tasks. The cost of duty or roster $r \in R$ is denoted by c_r and the penalty for not covering task $i \in I$ is denoted by p_i. Finally, b_i is the number of duties or rosters, which need to cover task $i \in I$. The SP on which the automatic planning part of the DSS is based is mathematically formulated as follows:

$$\text{(SP)} \quad \text{Minimize} \sum_{r \in R} c_r x_r + \sum_{i \in I} p_i s_i$$

subject to

$$\begin{aligned} &\sum_{r \in R} x_r + s_i = b_i && \forall i \in I, \\ &x_r \in \{0, 1\} && \forall r \in R, \\ &s_i \geqslant 0 && \forall i \in I. \end{aligned} \tag{1}$$

The objective usually comes down to primarily minimizing the number of uncovered tasks, and secondarily minimizing the total cost of the duties or rosters selected in the solution. Since Harmony CDR aims at supporting the planner, this primary goal implies that he or she has less work to do for covering all tasks. Constraints (1) are generalized set partitioning constraints, which ensure that each task is covered by at most b_i duties or rosters. In case of planning crew as a team, one duty is assigned to the entire team for the CRP. That is, for the right-hand side of constraint (1) b_i is equal to the number of crew in the team. Variables s_i, $i \in I$, are added to allow tasks to remain uncovered. Note that these variables can be continuous since the right-hand sides b_i, $i \in I$, as well as the x_r variables are integer. Variations for constraints (1) are possible by interchanging equality signs and inequality signs. For example, constraints (1) can model the possibility of crew deadhead-

ing by replacing $=$ with $\geqslant$. Additional constraints may be added for modeling global constraints, which deal with sets of duties or rosters, such as, e.g., a maximum fraction of duties with work time above a certain threshold. This model is based on the model proposed in (Ryan, 1992) for the CRP (see also (Gamache and Soumis, 1998; Gamache et al., 1999) for slight variations).

2.2. Solution approach

A disadvantage of the model above is a large number of feasible duties or rosters, which corresponds to a large number of columns. Therefore, in order to use the model in the solution approach, we need a column generation procedure. The state-of-the-art techniques in the automatic planning tool are related to the general framework in the context of time constrained routing and scheduling problems proposed by Desrosiers et al. (1995) and Desaulniers, Desrosiers and Solomon (2002). In particular, our solution approach consists of a branch-and-price algorithm for the SP. Branch-and-price extends branch-and-bound, because column generation is used throughout the entire branch-and-bound tree. Since the late eighties of the previous century several papers deal with column generation approaches for crew planning in passenger transportation (recent papers are (Desaulniers et al., 1997; Vance et al., 1997; Gamache et al., 1999)). The main steps of such an approach are:

Step 1. Read the input and choose the initial set of columns.

Step 2. Solve the LP relaxation with the current set of columns; this is usually called the *(restricted) master problem.*

Step 3. Column management: delete columns from the current LP model, and *price*: generate new columns with negative reduced costs and add them to the current LP model.
If new columns are found return to step 2.

Step 4. If stopping criteria are satisfied write the output and stop. Otherwise perform a new branch-and-bound step, that is:

- 4.1. *Select* a node to *branch* on,
- 4.2. Calculate the lower bounds of the two new nodes by solving the LP relaxations with the procedures in steps 2 and 3,
- 4.3. Perform fathoming rules; if an integer solution has been found that is better than the best integer solution so far, update the best integer solution,
- 4.4. Return to step 4.

The crucial aspects of this approach are *price*, *select* and *branch*, that is, how columns are generated in step 3, how nodes are selected, and which branching rule is used in step 4. In our implementation, nodes can be selected by depth-first-search, best-first-search, or a combination of these two. The system contains several algorithms for

generating columns based on one or more acyclic networks. For crew scheduling a single network is used, while for crew rostering a network is used for each crew. The networks are defined such that each node corresponds to a task and each arc corresponds to a feasible sequence of two tasks in one duty or roster. In addition, a source and sink arc are added to each network, denoting the start and end of a duty or roster. A path in a network corresponds to a feasible duty or roster if the path constraints are satisfied. See also (Desrochers et al., 1992) for an example of a similar network for crew scheduling. Paths through the networks are constructed by solving a resource constrained shortest path problem (see (Desrosiers et al., 1995)). The algorithms that we have implemented are a dynamic programming algorithm, a depth-first search algorithm, and an all-pairs shortest path algorithm (see (Freling, 1997)). The choice of the algorithm depends on the application considered, and combinations of two algorithms may also be used.

We have also implemented several branching rules. The branching rule for the exact branch-and-price algorithm consists of branching on the arcs in the underlying networks (see, e.g., (Ryan and Foster, 1981; Barnhart et al., 1998)). A branch consists of either forbidding or forcing one arc to be in the solution. The consequence for the child nodes is that several variables are fixed to zero, and that the corresponding networks are adjusted by deleting arcs to either force or forbid an arc to be in the solution. In case of forcing an arc to be in the solution, all other arcs leaving and entering the corresponding nodes are deleted.

This solution approach is very robust in the sense that both the constraints defining the feasibility of each duty or roster, and the constraints defining the feasibility of a set of columns can be easily incorporated without changing the basic structure of the algorithm (see section 3.2). In addition, for large scale problems several exact and heuristic techniques can be used in a straightforward manner to speed-up the algorithm (see section 4).

3. A decision support system for crew planning

In this section, we discuss the functionalities and the purpose of the DSS, the classification of the constraints and objectives in the system, and crew classes.

3.1. Functionality and purpose of the DSS

The Harmony CDR system supports the entire planning process, which is divided into long term planning, short term planning (rostering and operations control), realization and evaluation. This is depicted in figure 1.

Starting points are defined based on long term analyses and company rules. For example, a starting point can be the desired service level, the desired number of crew, training needs, holidays, and other specific issues that influence the crew availability. The long term planning concerns planning for an entire season, that is based on the defined starting points. The system helps management to determine:

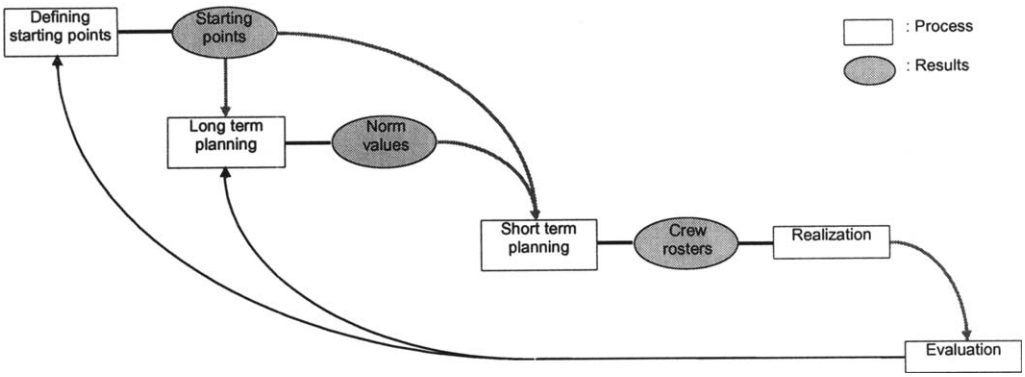

Figure 1. The planning process.

- The capacity needed to perform the tasks.
- The permanent and temporarily staffing levels needed to meet the required capacity.
- Allowed vacations, standbys required, etc. for the given period.

The long term planning delivers several norm figures for use in the planning department. Examples are the distribution of crew over depots and parameters regarding crew satisfaction, such as a maximum average working time. When the norm figures are available, a planner can start creating rosters. Of course these norm figures influence the planner's freedom of movement. An example of an application of the short-term module is to determine daily duties first by solving the CSP seven times for each day of the week, and determining weekly rosters afterwards by solving the CRP. After the roster has been published, several changes can occur due to, for example, sickness of crew, schedule changes, delays, etc. The system supports the planner during the coordination stage and the operations control stage. Based on reports and a user-friendly planning board, the planner will be able to handle the operational changes.

Once a specific duty has been performed, corresponding data are saved into Harmony CDR's database. For example, this concerns crew notification data such as sign-in/sign-out. To complete the planning circle, the system supports the planner to help making an evaluation of the realization.

The best rosters are those that take into account management as well as crew wishes. But these are often conflicting, so it is very difficult to create a roster manually that will keep both parties satisfied. The automatic planning tool facilitates the planner in creating a roster that satisfies both economic and social criteria (see section 3.2). The system concept relies on the possibility of calculating several alternatives within a relatively short time (what-if analysis). What-if analysis can be used on every planning process level supported by the system. For example, what-if analysis can be used to analyze the effects of certain policies, to support negotiations with respect to legal regulations, to make a capacity plan, or to select the best alternative out of several generated rosters.

3.2. Classification of constraint and objective types in the DSS

The duties and rosters that are generated by the automatic planning tool have to meet certain labor laws and agreements in order to be feasible. Assume that we have an underlying network representation as proposed in section 2.2. Recall that the network representation is such that each node corresponds to a task and each arc corresponds to a feasible sequence of two tasks in one duty or roster. A path in a network corresponds to a feasible duty or roster if the path constraints are satisfied. We distinguish three levels of constraints:

1. *Coupling constraints.* This concerns rules that relate to the whole set of selected duties or rosters. Such a constraint could specify, for example, that at most 5% of all selected duties have durations longer than 9 hours. These constraints are added to the master problem.
2. *Path feasibility constraints.* Rules at this level determine the feasibility of a duty or roster (a path in a network representation). For example, the maximum length of a duty is 9.5 hours. Usually, most constraints are at this level.
3. *Node/arc feasibility constraints.* A constraint at this level determines if a particular task (a node in a network representation) can be assigned to a certain crew due to licenses, etc. or if two tasks (or duties) can be assigned consecutively to the same duty or roster (an arc in a network representation). For example, an arc from task i to task j exists if task j starts after task i has finished and a short buffer time has passed.

A similar classification holds for other network representations. Sometimes the network representation needs to be modified in order to be able to check constraints efficiently. For example, several home bases may exist, and each duty or roster must start and end at the same home base. In that case, it may be better to use one network for each home base. In case of one network for each crew, this problem would be solved if each crew has a unique home base. However, in one of the airline applications we worked on (see section 5), some crew could be assigned to two home bases because they lived somewhere in between.

For the classification of objectives we discern three types of possibly conflicting objectives:

1. *Efficiency*: cost minimization with respect to the number of uncovered tasks and the number of required duties or crew; other cost factors may be the number of layovers and the total time between tasks. Uncovered tasks need to be planned manually afterwards and might require hiring additional crew.
2. *Welfare*: the workload of the rosters should be equally spread among the crew, and requests should be granted equally as much as possible.
3. *Robustness*: duties or rosters must be robust with respect to delays, for example. For instance, this can be accomplished by imposing a maximum number of vehicle changes in a duty, or by requiring minimum transfer times.

In an operational setting, an important objective typically is to balance the workload among the crew. We balance workload as a weighted three-period average in our system, where a period has the same length as the period of planning. From the profile of each crew, we know the workload of the crew in the past two periods. This workload includes work time as well as penalties for undesired characteristics, such as layovers, heavy-duty sequences and standby duties. Together with the tasks that will be assigned, we are able to compute a weighted average workload over the two periods history and the period of planning during the preprocessing phase. Workload balancing is activated during the optimization process by penalizing rosters where the three-period, weighted workload exceeds a threshold based on the weighted average workload of these three periods. A similar classification as for the constraints is also possible: some objectives have an effect on a set of duties or rosters, some on paths, and some on nodes and/or arcs. Note that the constraints and objectives that need to be checked at each level depend on the application, the client, and the planning horizon. For example, for the CSP on a daily basis only efficiency and robustness are relevant. Or, for the short term CRP, crew welfare and robustness could be more important than efficiency if all crew receive fixed salaries anyway.

3.3. Crew classes

Usually, in the airline and railway industry several crew classes like cockpit and cabin crews need to be planned. Cockpit and cabin crews usually consist of different crew functions, for instance, pilots and co-pilots for a cockpit crew. Although crew can be planned for each function separately, it may be efficient to use crew teams of, for example, cockpit crew only or of both cockpit and cabin crew together. Often, different duty functions can be assigned to one crew function, for example, a pilot can do both a pilot duty and a co-pilot duty. In that case, and if crews do not need to be planned as teams, a logical decomposition is to solve the co-pilot CRP first and all the duties that are uncovered are added to the pilot problem. Another example of different crew classes would be train drivers and guards, see (Fischetti and Kroon, 2000). The context of crew planning for urban transit has been discussed in detail in (Odoni and Rousseau, 1994). Furthermore, Desaulniers et al. (1998a, b) have provided an overview of different crew planning problems.

4. Implementation issues

As mentioned in the introduction, implementing a branch-and-price algorithm is not an easy task, let alone implementing a flexible tool, suitable for a variety of applications. However, in the literature little or no attention has been paid to the implementation of such an algorithm. Therefore, we provide some insights into our implementation, and discuss some difficulties we ran into and the corresponding solutions we came up with.

4.1. Abstract implementation

The framework is implemented in the C++ programming language, making fully use of its object-oriented nature. Figure 2 shows an overview of the implementation.

The optimization coordinator steers the various calls to the branch-and-price algorithm, like solving the CSP first and then the CRP, and solving the problem for different crew functions. There is an abstract interface with application specific information, that is, rules checking input and output translator (via file interface) and parameter tuning. There is also an abstract interface with the linear programming solver (currently CPLEX 6.6, XPRESS 12.1, and a subgradient algorithm), and an abstract interface with the pricing algorithm.

An important part in such an abstract implementation is the rules checker, which also incorporates the functionality to calculate (elements of) the objective function. Recall that three levels of constraints and objectives can be defined. The link between the rules checker and the integer programming (IP) solver in figure 2, corresponds to the checking of the coupling constraints of the SP. The path constraints are checked during the network algorithm, and the node/arc constraints are checked during the construction of the network(s). Also note that the constraints and objectives that need to be checked at each level, depend on the application and the planning horizon. For computational efficiency it is crucial that the path constraints can be checked as fast as possible. In our opinion, a column generation algorithm only works well if the path constraints are checked in sequence, that is, when building a path, each time a node is added to the path, it is not necessary to check the entire path so far. This is achieved by keeping constraint information of the path so far. An exception may be that one or two rules are checked at the end, when a complete path is found. Thus the feasibility of a path up to a certain node (no matter which algorithm is used) is checked by combining the constraint information of the path so far with the relevant information of adding the node. For this purpose, two vectors are used: a consumption vector with the *consumption* of each resource for a path up to a node, and an arc vector with the consumption of each resource for an arc (incorporating the addition of the node to which the arc points). The network algorithm

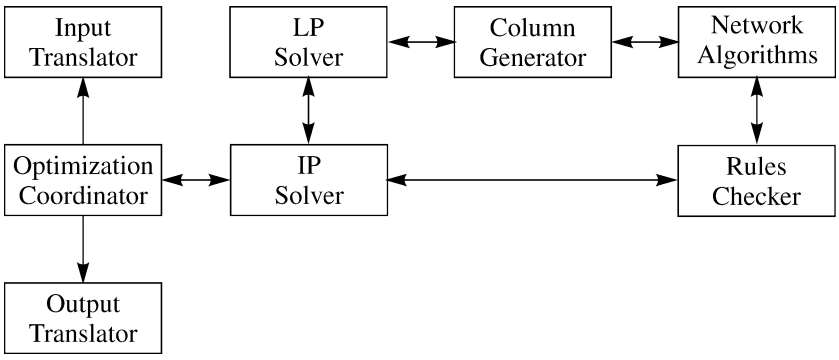

Figure 2. Abstract implementation.

is not aware of details on these general vectors. These details are only known at the rules checker level, which keeps the implementation of the network algorithm general and abstract.

4.2. Efficient use of computer memory

The memory usage of a branch-and-price algorithm grows very fast with the size of the problem. To give an idea about the memory usage, we use a CRP example where a dynamic programming algorithm is used for the pricing problem. The larger part of memory is used by four categories:

1. Size of the network for each crew (mainly determined by the size of the arc vector).
2. Size of the LP model.
3. Size of the branch-and-bound tree.
4. State space for the dynamic programming algorithm.

Advanced data structures are necessary to prevent the memory requirements from exploding to an unmanageable size, while keeping run times as low as possible. For the first category we reduced the need for computer memory by sharing nodes of networks with crew. For a specific crew, the network differs with the network of another crew by linking only those nodes and arcs in the network that are relevant for that particular crew member. For an application consisting of 109 crew, a memory reduction of a factor 6 is obtained (from 680 to 112 Mb), compared to creating a complete new network for each crew.

Of course, it is undesired to keep the entire content of the LP model at each node of the branch-and-bound tree since it results in huge memory requirements as well as a lot of duplicate information. Therefore, we keep the entire LP model at the root node of the tree, and store the generated as well as deleted columns at each node. Furthermore, for combinations of specific branching strategies and column management strategies, more efficient book-keeping tactics can be implemented.

The branch-and-bound tree is implemented as a heap, where active nodes are kept. This datastructure particularly works well in combination with a depth-first-search strategy. Such a strategy can also reduce memory usage, because the number of open problems remains relatively small.

The state space for the dynamic programming algorithm consists of a linked list of paths at each node. The memory of these paths is never released during the algorithm to prevent wastage of computer time due to allocating and deleting memory.

4.3. Exact strategies for computer time reduction

Here, we provide a list of the exact strategies that we have implemented and tested.

4.3.1. Efficient column management

The more columns are added to the master problem in each iteration, the less column generation iterations are necessary but the more time it takes to solve each LP problem. Therefore, experimentation is necessary to determine the right column management strategy. A discussion on column management strategies and related topics can be found in (Bixby et al., 1992; Kohl, 1995; Freling, 1997). By experimenting with various applications, we found a strategy that works well in general. The idea is to generate a relatively high number of columns, add them to the LP problem, and delete a large percentage of those columns after solving the LP problem. In that way many columns are considered but the LP problem does not grow too fast. The number of columns that can be generated depends on the pricing algorithm used. For all the algorithms we used in our implementation it is possible to influence the number of columns generated. The strategy for deleting columns after solving the LP problem is as follows (see also (Freling, 1997)): let q be the number of positive reduced cost columns after solving the LP problem, that are selected from those columns that are added to the master problem just before solving the LP problem. Then, delete $\lfloor 0.9q \rfloor$ of these columns. The columns to be deleted are the columns with the largest reduced cost.

Our computational tests confirmed that it is very beneficial to use partial pricing (see also (Gamache et al., 1999)) for the CRP. In each iteration, we generate columns once either negative reduced columns have been found for p_1 crew, or the pricing problem has been solved for all crew. The value of p_1 is set to $\max(5, \lfloor \#\text{crew}/20 \rfloor)$ in our implementation. If p_2 is the number of negative reduced cost columns generated, then $\min[r, p_2]$ columns are added to the master problem. The values for r and p_1 may also be adjusted dynamically during the algorithm and need to be tuned for each particular application. A low value of r implies that little columns will be added in each iteration of column generation. The choice of value for r is typically based on experience.

4.3.2. Temporary relaxation of the pricing problem

The purpose of the pricing problem is twofold: generating 'good' columns, and providing a criterion for convergence of the column generation algorithm. For the second purpose, the pricing algorithm needs to be exact, for the first purpose not. Therefore, it may be efficient to solve the pricing problem heuristically until some point and then change to an exact algorithm (see also (Sol, 1994; Gamache et al., 1999)).

We have tested a strategy called *dynamic network size*. We start with a sparse subnetwork by leaving out arcs, and update the network size iteratively until the original network size is obtained. In each iteration of the column generation procedure, the arcs in the network are sorted increasing reduced cost and the set of arcs in the subnetwork is selected. Thus, when selecting the subnetwork in each iteration, we take the subset of arcs with the lowest reduced cost out of each node. The reduced cost of an arc is defined as the cost of an arc (which includes the cost of the node it points to) minus the dual value of the task corresponding to the node it points to. The maximum number of arcs to be selected in each iteration is increased during the column generation procedure.

4.4. Heuristic strategies for computing time reduction

The branch-and-price algorithm is very well suited to add heuristic features that speed up solution time considerably while the solution remains close to optimality. See, for example, (Gamache et al., 1999). Here we discuss several of such features.

4.4.1. Fixed relaxation of the pricing problem

When using dynamic programming to solve the pricing problem, some constraints or criteria can cause the state space to grow very fast. Let s and t be the source and the sink of a network, respectively. The size of the state space depends on the number of paths from s to each node, that is, the number *of partial paths*. To reduce the state space, dominance criteria are used to remove partial paths that will not lead to the shortest s–i path. Let $P_i(s)$ and $Q_i(s)$ denote two partial paths from s to node i. Let $U_i(t)$ denote a partial path from node i to the sink t. Then $P_i(s)$ dominates $Q_i(s)$ if the following two conditions are satisfied:

1. The (reduced) cost of $P_i(s)$ is less or equal to the (reduced) cost of $Q_i(s)$; and
2. If $P_i(s) \cup U_i(t)$ is infeasible, then $Q_i(s) \cup U_i(t)$ is also infeasible for all $U_i(t)$.

The first condition is only valid when the cost function is (monotonically) non-decreasing when a partial path is extended with a node. The second condition is only valid when an ordering exists for all the constraints. For example, in case of a maximum work time, the ordering follows from the fact that less work time is always better. Thus, if maximum work time is the only constraint, $P_i(s)$ dominates $Q_i(s)$ if both the cost and the work time of $P_i(s)$ are less or equal to the cost and the work time of $Q_i(s)$. In case we also have a minimum work time constraint, the dominance rule above can only be applied when both partial paths satisfy this constraint, thus resulting in a larger state space. Finally, suppose we have an objective that the work time should be as close as possible to a certain fixed value, that is, the cost function is not (monotonically) increasing. Then, dominance can only be applied once both paths considered have exceeded the fixed value.

In practice, we have encountered several of these types of constraints and objectives that cause the state space to grow very fast. But in none of those cases the user of the system had a problem when we modified the constraint or objective slightly in order to deal with them in a computationally more attractive way. For example, the minimum work time is relaxed, and only when the path is completed (that is, it reaches t) a penalty is added when the work time is below its minimum. In case the objective is that the work time should be as close as possible to a fixed value, costs are only involved once the work time exceeds the fixed value. Thus, dominance can be applied below that value as well. Experience shows that because the intention of the constraint is to balance work time among crew, this can still be achieved approximately by setting the fixed value a little lower. Because a nonlinear penalty function is used (the higher the work time the faster the penalty increases), the work time is pushed down towards this value from above.

4.5. Other strategies that can be used

In this section we will discuss a list of other strategies that are relevant for implementing a branch-and-price algorithm. We suggest by no means that this list is exhaustive, but merely indicate strategies that can be of a practical interest, when one wants to implement such a system.

1. Stabilized column generation, see (Du Merle et al., 1999; Desrosiers et al., 2001). Here, one tries to converge the dual variables smoothly, which is typically not the case.
2. Lower bound approximation, see (Freling, 1997), to handle stalling and tailing off problems during the column generation process, without violating a predefined optimality tolerance.
3. Generating upper bounds. During the branch-and-price algorithm it typically pays off to find better upper bounds in a node of the tree using intelligent heuristics. This way, nodes will be fathomed earlier, which will typically improve runtime. For example, if our branching strategy is branching on variables, it might pay off to fix several variables at the same node of the tree (see also (Gamache et al., 1999)).
4. Grouping crew with similar characteristics. When employing partial pricing, it is beneficial to group crew with similar characteristics. This way, columns are generated for the entire group at once and later on divided among the crew.
5. Stop-criteria used in the algorithm. In practical applications near optimal solutions in short computation times are desired. Therefore it is important to have good stop-criteria. Such criteria can, e.g., be based on the improvement in the lower bound, or a maximum number of iterations of column generation in the branch-and-bound tree.
6. Network reduction techniques for the dynamic programming problem in the pricing problem. The rules of a certain application may have impact on the network that is used for the dynamic programming problem. Taking these rules into account already at the network design phase, if possible, is more efficient than taking them into account during the construction of the columns, see (Freling, Lentink and Odijk, 2001) for more information.
7. In the network used for dynamic programming arcs could be left out heuristically. It is very well possible that certain combinations of tasks result in highly undesired rosters or duties, e.g., a duty with a lot of idle time between two tasks or flights. Forbidding such arcs will result in smaller networks, and can speed up dynamic programming at the cost of the possibility to loose global optimality.

Additional information on acceleration techniques for column generation algorithms can be found in (Desaulniers, Desrosiers and Solomon, 2002; Freling, 1997).

5. Applications of the automatic crew planning module

The automatic planning module of the DSS has been implemented for several applications. In this section, we briefly consider a charter airline and two railway applications

and we consider a case study for a European regional airline company in more detail in section 5.2. All computational results were carried out on a PC with a Pentium II 400 Mhz processor and 128 Mb internal memory.

5.1. Example applications

In (Freling, Lentink and Odijk, 2001), the authors consider a case study for the Dutch Railways in detail. We use a heuristic branch-and-price algorithm to solve CSPs with at most 1114 tasks. The average duration of a task is approximately 35 minutes. Furthermore, the case has 12 stations, including 7 endpoints and two stations where trains can be split and combined. The objective is to minimize the number of crew, to minimize the number of transfers, as well as minimizing crew deadheading. This largest instance is solved in about 32 minutes with a gap of 2.32%. The pricing problem is solved exactly by dynamic programming. The heuristic feature is in the branching step by only performing a few iterations of column generation when solving the LP problem in other nodes than the root node, and to try to find a good feasible solution as quickly as possible using a depth-first-search strategy during branch-and-price. The algorithm stops once a feasible solution is found. The extension by fixing several variables to one at once in each node of the tree has also been tested, and resulted in a solution with 3.69% gap in about 11 minutes on the same instance.

One of the users of the DSS is the European company RailRest that operates the catering of the Thalys, a high-speed train between major cities in the western part of Europe. The system is used for planning catering personnel working on trains. Duties are constructed manually in the DSS, and the automatic planning module is used for a monthly CRP. Because run time is considered as important as the quality of the solution, the branch-and-price algorithm is a bit heavy for this type of problem. Furthermore, the problem size is about 126 crew and about 2200 duties, which is relatively large. An additional complication is the objective function. To minimize the number of uncovered duties is no longer the primary objective. This is because balancing the workload is a company rule, and is considered more important. This is achieved by using a planning period of one month in the procedure regarding workload balancing in section 3.2. The uncovered tasks are assigned manually afterwards to external personnel. Even when sequentially solving the problem week by week and using several heuristic features the run time is still about 2 hours and 40 minutes, which is considered too long. Therefore, the weekly problem is split in four smaller problems by logically dividing the crews and the duties in two subproblems. The four crew classes are obtained as follows: one type for crews with a 38-hour per week contract, two types of crews with a 32-hour contract, and one type for crew with a 20 and 25-hour contract. Then, the total run time is about 80 minutes. This is acceptable, but we have also developed a simple algorithm where we sequentially solve an assignment problem for each pair of days. The quality of this solution is generally acceptable, and the run time is only about 10–15 minutes in total.

5.2. Application to crew restoring for a regional airline

The case study for a European regional airline company is interesting from both a practical and a computational point of view, because the laws, regulations and company rules are very complex. The aim of the study was to automatically build efficient weekly rosters for cockpit crew. Distances are relatively short since the company operates domestic flights and flights between the home country and neighboring countries. Tasks that are input for the daily CSP consist mainly of couples of flights and a number of single flights. The network of this company resembles a star network and typically one flies from his home base to an outstation and returns to his home base. The case has approximately 500 of these couples as input. In addition, a duty consists of 2 to 3 of these couples. After solving the daily CSPs, the input of the CRP is about 280 duties and about 55 crew. Because co-pilots and pilots are planned separately, this resulted in two problems of about half the size. Other characteristics are:

- Multiple aircraft types,
- A highly connected network of flights,
- Several crew specific characteristics, among which fixed tasks, crew availability, past workload, and licenses for aircraft types and for airports,
- Multiple home bases per crew,
- A night stop (layover) off home base is allowed,
- No deadheading on flights, only by car or bus rides between home bases.

Each crew has a pre-defined *profile* that forms part of the input of the CRP module. Such a profile consists mainly of fixed tasks and already granted requests. However, most requests are not granted a priori, but are assigned during the optimization process.

5.2.1. Constraints and objectives

A complicating company constraint is that each crew needs to end a duty at the same home base where she/he started. In case of layovers, the crew needs to end at the same home base it started its off home base work period. This is incorporated in the system by keeping an extra variable in the consumption vector that denotes the start home base.

There are too many laws and regulations to mention them all here. Some examples of complex constraints are:

- In every period of 7 days at least 36 hours rest period, or in every period of 10 days at least 48 hours rest period.
- Between each two daily duties at least 18 hours rest, and if violated, a 15-day period starts with variable minimum allowed cumulative rest. This period ends once the cumulative rest exceeds a norm value.
- The maximum allowed working time is dependent on the number of landings, on the start time of the daily duty, and, in case the minimum rest time is violated, also on the rest period and on the working time of the previous day.

The first constraint can be interpreted in several ways. Contacting the Labor Inspection Office that designed the rule did not clarify it because the person who designed the rule does not work there anymore and there is nobody else who can resolve the ambiguities. So, we chose the interpretation that the company was used to, which is relatively easy to implement in a column generation context. We keep two variables (resource consumptions), namely, the number of days since a 36-hour rest period and the number of days since a 48-hour rest period. A path cannot be extended once the extension incurs that both rules will be violated. That is, in the past 7 days no 36 hour rest period occurred and in the past 10 days no 48 hour rest period occurred.

Because we are dealing with weekly rosters in an operational setting, minimizing costs is not a main objective. After the minimization of the number of unassigned flights, the most important objective is to balance the workload among the crew. From the profile of each crew, we know the workload of the crew in the past two weeks. This workload includes work time as well as penalties for undesired characteristics, such as layovers, heavy-duty sequence and standby duties. Together with the flights that will be assigned, we are able to compute a weighted average workload over the last three weeks during the preprocessing phase. Workload balancing is activated during the optimization process by penalizing rosters where the three-week, weighted workload exceeds a threshold based on the weighted average workload of these three weeks.

5.2.2. Algorithm

We used the same branch-and-price algorithm for the CSP and the CRP, that is, with the same parameter values. The CSP problems are relatively small and can be solved within a second. Although the size of the CRP problem is moderate (about 250 duties and 35 crew), it is still a complex problem because of the highly complex objectives and constraints as well as the high interconnectivity of tasks. We use dynamic programming for column generation. The only heuristic feature of the algorithm is that not all constraints are incorporated in the dominance testing, that is, optimal paths could be dominated. For the tests we performed, the results are optimal or close to optimal, while the memory usage and computation time is reduced significantly. We used the set partitioning version of our algorithm, because set covering is not feasible. Overcovers cannot always be removed afterwards, since that may cause duties to start and end at different locations. We also implemented a version where rosters are built directly from flights without first solving the CSP. This version also uses the framework of figure 2. However, the input now consists of the tasks for the entire week, instead of tasks for one day (for the CSPs) or duties for each day (for the CRP). Furthermore, we need to integrate the CSP rules and the CRP rules in the Rules Checker.

5.2.3. Computational results

In table 1 below we compare the results for the regular sequential approach of first scheduling then rostering with the integrated approach. For the sake of easy comparison, the objective here is to minimize the number of uncovered tasks primarily, and the number of crew secondarily.

Table 1
Integrated scheduling and rostering.

Data set	# tasks	First scheduling			Integrated		
		CPU (s)	# crew members	# not scheduled	CPU (s)	# crew members	# not scheduled
A	196	37	32	8	96	32	3
B	244	359	39	0	722	39	0
C	521	41	53	17	96 657	51	0

Table 2
Branch-and-bound vs. branch-and-price.

Data set	Algorithm	CPU (s)	Duality gap (%)		# crew members	# not scheduled	# branches	
			co-pilot	pilot			co-pilot	pilot
A	B&B	61	11	5	35	8	200	200
A	B&P	37	0	0	32	8	17	3
B	B&B	124	15	17	43	0	200	200
B	B&P	359	0	0	39	0	28	40
C	B&B	47	7	44	54	24	200	200
C	B&P	41	0	0	53	17	12	13

The second column in the table shows the number of tasks, the third to fifth column show, respectively, the run time, the number of crew, and the number of tasks that have no roster for the sequential approach. The sixth to the eighth column show the same numbers for the integrated approach. For data set *B* there is no difference in solution quality, but for data sets *A* and *C* there is a large difference. In data set *A*, 5 tasks that are not assigned in the sequential approach can be assigned in the integrated approach, and the run time is about 3 times higher. In data set *C*, all 17 unassigned tasks of the sequential method could be assigned in the integrated method while using two crews less. The run time explodes from 41 second to almost 27 hours. The conclusion is that the way the duties are built can have an enormous influence on the quality of the roster solution and the run time.

In table 2, we show the results of a comparison of the branch-and-price algorithm (B&P) with an earlier branch-and-bound version of the system (B&B) where we used column generation only in the root node. The same objective is used as in the previous table.

As mentioned in section 5.1, this earlier version produced good quality solutions for an application of crew rostering for a Dutch charter airline. However, as can be seen from the table, the solutions of the B&B version can produce a gap of up to 44%, which is highly unacceptable. Note the number of branches in the last two columns. The maximum number of branches for the B&B version is 200, but the solutions do not improve significantly when increasing this number.

6. Summary

This paper has discussed a decision support system for airline and railway crew planning. The focus is not on the mathematical part, since the basic ideas have been reported in other papers, but on the system, the implementation and the applications. We give insight in several implementation issues that usually come up when implementing a branch-and-price algorithm for a practical application. When looking at the development of the algorithms in the system, it is interesting to note that for the first airline application before 1996, a simple day-by-day heuristic was not appropriate. On the contrary, in the last railway application a similar simple heuristic turned out to be more appropriate. The cause can be found in several factors, among which are the type of constraints and objectives, and the definition of the duties. In the case of regular duties of similar length a heuristic seems to have more success as compared to a more exact method. Thus, when working in practice for each application the best method needs to be determined. Finally, we can conclude for the comparison with the integrated scheduling and rostering that the way duties are built can have a huge impact on the quality of the rosters at the cost of a huge increase in computing time.

Acknowledgments

The authors like to thank Daan Ament of ORTEC for his valuable comments. Furthermore, we would like to thank the referees for their useful comments and remarks.

References

Andersson, E., E. Housos, N. Kohl, and U. Wedelin. (1998). "Crew Pairing Optimization." In G. Yu (ed.), *Operations Research in the Airline Industry*. Dordrecht: Kluwer Academic.

Barnhart, C., E.L. Johnson, G.L. Nemhauser, M.W.P. Savelsbergh, and P.H. Vance. (1998). "Branch-and-Price: Column Generation for Solving Huge Integer Programs." *Operations Research* 46, 316–329.

Bixby, R., J. Gregory, I. Lustig, R. Marsten, and D. Shanno. (1992). "Very Large-Scale Linear Programming: A Case Study in Combining Interior Point and Simplex Methods." *Operations Research* 40, 885–897.

Bodin, L., B. Golden, A. Assad, and M. Ball. (1983). "Routing and Scheduling of Vehicles and Crews: The State of the Art." *Computers and Operations Research* 10(2), 63–211.

Caprara, A., M. Monaci, and P. Toth. (2001). "A Global Method for Crew Planning in Railway Applications." In S. Voß and J.R. Daduna (eds.), *Computer-Aided Scheduling of Public Transport*, Lecture Notes in Economics and Mathematical Systems, Vol. 505. Berlin: Springer.

Day, P.R. and D.M. Ryan. (1997). "Flight Attendant Rostering for Short-Haul Airline Operations." *Operations Research* 45, 649–661.

Desaulniers, G., J. Desrosiers, Y. Dumas, S. Marc, B. Rioux, M.M. Solomon, and F. Soumis. (1997). "Crew Pairing at Air France." *European Journal Operational Research* 97(2), 245–259.

Desaulniers, G., J. Desrosiers, M. Gamache, and F. Soumis. (1998a). "Crew Scheduling in Air Transportation." In T.G. Crainic and G. Laporte (eds.), *Fleet Management and Logistics.* Boston: Kluwer Academic.

Desaulniers, G., J. Desrosiers, I. loachim, M.M. Solomon, F. Soumis, and D. Villeneuve. (1998b). "A Unified Framework for Deterministic Time Constrained Vehicle Routing and Crew Scheduling Problems." In T.G. Crainic and G. Laporte (eds.), *Fleet Management and Logistics*. Norwell: Kluwer.

Desaulniers, G., J. Desrosiers, and M.M. Solomon. (2002). "Accelerating Strategies in Column Generation Methods for Vehicle Routing and Crew Scheduling Problems." In C.C. Ribeiro and P. Hansen (eds.), *Essays and Surveys in Metaheuristics*. Boston: Kluwer.

Desrochers, M., J. Gilbert, M. Sauve, and F. Soumis. (1992). "Subproblem Modeling in a Column Generation Approach to the Urban Crew Scheduling Problem." In M. Desrochers and J.M. Rousseau (eds.), *Computer-Aided Transit Scheduling: Proceedings of the 5th International Workshop*. Berlin: Springer.

Desrosiers, J., Y. Dumas, M.M. Solomon, and F. Soumis. (1995). "Time Constrained Routing and Scheduling". In M.O. Ball, T.L. Magnanti, C.L. Monma, and G.L. Nemhauser (eds.), *Handbooks in Operations Research and Management Science*, Vol. 8: *Network Routing*. Amsterdam: North-Holland.

Desrosiers, J., A. Lasry, D. Mclnnis, M.M. Solomon, and F. Soumis. (2000). "Air Transat Uses ALTITUDE to Manage Its Aircraft Routing, Crew Pairing, and Work Assignment." *Interfaces* 30(2), 41–53.

Desrosiers, J., M.M. Solomon, D. Villeneuve, and H. Ben Amor. (2001). "Stabilized Column Generation for Crew Scheduling Problems." *Presentation at TRISTAN IV Conference*, Azores Islands.

Du Merle, O., D. Villeneuve, J. Desrosiers, and P. Hansen. (1999). "Stabilized Column Generation." *Discrete Mathematics* 194, 229–237.

Fischetti, M. and L.G. Kroon. (2000). "Solving Large-Scale Crew Scheduling Problems at the Dutch Railways." In S. Voß and J.R. Daduna (eds.), *Computer-Aided Scheduling of Public Transport*, Lecture Notes in Economics and Mathematical Systems, Vol. 505. Berlin: Springer.

Freling, R. (1997). "Models and Techniques for Integrating Vehicle and Crew Scheduling." Ph.D. thesis, Tinbergen Institute, Erasmus University, Rotterdam.

Freling, R., R.M. Lentink, and M.A. Odijk. (2001). "Scheduling Train Crews: A Case Study for the Dutch Railways." In S. Voß and J.R. Daduna (eds.), *Computer-Aided Scheduling of Public Transport*, Lecture Notes in Economics and Mathematical Systems, Vol. 505. Berlin: Springer.

Gamache, M. and F. Soumis. (1998). "A Method for Optimally Solving the Rostering Problem." In G. Yu (ed.), *Operations Research in the Airline Industry.* Dordrecht: Kluwer Academic.

Gamache, M., F. Soumis, G. Marquis, and J. Desrosiers. (1999). "A Column Generation Approach for Large-Scale Aircrew Rostering Problems." *Operations Research* 47, 247–263.

Gamache, M., F. Soumis, D. Villeneuve, J. Desrosiers, and E. Gélinas (1998). "The Preferential Bidding System at Air Canada." *Transportation Science* 32, 246–255.

Kohl, N. (1995). "Exact Methods for Time Constrained Routing and Scheduling Problems." Ph.D. thesis, Technical University of Denmark.

Nicoletti, B. (1975). "Automatic Crew Rostering." *Transportation Science* 9, 33–42.

Odoni, A.R. and J.M. Rousseau. (1994). "Models in Urban and Air Transportation." In S.M. Pollock, M.H. Rothkopf, and A. Barnett (eds.), *Handbooks in Operations Research and Management Science*, Vol. 6: *Operations Research and the Public Sector*. Amsterdam: North-Holland.

Ryan, D.M. (1992). "The Solution of Massive Generalized Set Partitioning Problems in Aircrew Rostering." *Operations Research* 43, 459–467.

Ryan, D.M. (2000). "Optimization Earns Its Wings." *OR/MS Today* 27(2), 26–30.

Ryan, D.M. and Foster. (1981). "An Integer Programming Approach to Scheduling." In A. Wren (ed.), *Computer-Aided Scheduling of Public Transport*, Lecture Notes in Economics and Mathematical Systems. Amsterdam: North Holland.

Sol, M. (1994). "Column Generation Techniques for Pickup and Delivery Problems." Ph.D. thesis, Eindhoven University of Technology.

Vance, P.H., A. Atamtiirk, C. Barnhart, E. Gelman, E.L. Johnson, A. Krishna, D. Mahidhara, G.L. Nemhauser, and R. Rebello. (1997). "A Heuristic Branch-and-Price Approach for the Airline Crew Pairing Problem." Technical Report, Georgia Institute of Technology.

Warburton, A. (1987). "Approximation of Pareto Optima in Multiobjective, Shortest Path Problems." *Operations Research* 35, 70–79.