Alcom-FT Technical Report Series
ALCOMFT-TR-01-91

Constraint Programming Based Column Generation for Crew Assignment*

Torsten Fahle† Ulrich Junker‡ Stefan E. Karisch§
Niklas Kohl§ Meinolf Sellmann† Bo Vaaben¶

Abstract

Airline crew assignment problems are large-scale optimization problems which can be adequately solved by column generation. The subproblem is typically a so-called constrained shortest path problem and solved by dynamic programming. However, complex airline regulations arising frequently in European airlines cannot be expressed entirely in this framework and limit the use of pure column generation. In this paper, we formulate the subproblem as a constraint satisfaction problem, thus gaining high expressiveness. Each airline regulation is encoded by one or several constraints. An additional constraint which encapsulates a shortest path algorithm for generating columns with negative reduced costs is introduced. This constraint reduces the search space of the subproblem significantly. Resulting domain reductions are propagated to the other constraints which additionally reduces the search space. Numerical results based on data of a large European airline are presented and demonstrate the potential of our approach.

Keywords: airline crew assignment, constraint satisfaction, column generation, shortest path constraint, hybrid OR/CP methods

---

*This work was partially supported by the IST Programme of the EU under contract number IST-1999-14186 (ALCOM-FT).

†University of Paderborn, Department of Mathematics and Computer Science, Fürstenallee 11, D-33102 Paderborn, Germany, `tef@upb.de` and `sello@upb.de`

‡ILOG S.A., 1681, route des Dolines, F-06560 Valbonne, France, `junker@ilog.fr`

§Carmen Systems AB, Odinsgatan 9, S-41103 Gothenburg, Sweden, `stefank@carmen.se` and `niklas@carmen.se`

¶Technical University of Denmark, Department of Mathematical Modeling, Building 321, DK-2800 Lyngby, Denmark, `bo.vaaben@sas.dk`

1 Introduction

1.1 The Crew Assignment Problem

Routing and scheduling of crews and equipment in large public transportation networks such as airlines, railways, and bus companies has been a major field for optimization for a long time. Different planning problems such as fleet assignment, aircraft routing, and crew scheduling have been addressed by numerous research papers. A recent book of Yu [32] and the article of Rushmeier et al. [29] give examples of problems and solution approaches in the airline industry.

In the airline crew scheduling problem a set of basic activities or tasks[1] – typically, to fly an aircraft from A to B – has to be assigned to a set of crew members such that all basic activities are covered. Additionally, complex rules and regulations coming from legislation and contractual agreements have to be met by the solutions. Different objectives like minimizing overall costs and maximizing crew satisfaction are of interest. The latter objective is chosen in so-called preferential bidding systems where crew members are allowed to express their likes and dislikes for certain activities (see, e.g., [17]).

In larger airlines the crew scheduling problem is decomposed into a crew pairing and a crew assignment (or rostering) problem. In the crew pairing problem the basic activities, *flight legs* (flight without stopover), are grouped into so-called pairings. A pairing is a sequence of flight legs usually starting and ending at the same home base and is considered one single "piece" of work. A pairing corresponds to one or more days of work and will be assigned as a whole to one or more named individuals. However, in the crew pairing problem the pairings remain anonymous. Pairings have to respect certain rules, e.g., connection time between flight legs and rest time between duty days. The crew pairing problem has been studied by several authors (see, e.g., [1, 3, 11, 16]).

The topic of this paper is the *crew assignment problem* (CAP). In the CAP the pairings together with other activities such as ground duties, reserve duties, and off-duty blocks must be assigned to named individuals. The outcome of this is a so-called *roster* for each crew member. Rosters must respect rules and regulations additionally to those considered in the pairing problem. These rules can be classified into *single crew member rules* and *multiple crew member rules.* The rules of the first class only influence the legality of a single roster, whereas those of the second class express regulations on combinations of rosters or crew members. Crew assignment is considered to be a large scale optimization problem, as in practice several thousand pairings and other activities have to be assigned to hundreds or possibly even thousands of crew members. The number of rules and regulations which must be respected can be up to one hundred and the number of possible rosters is gigantic.

1.2 Current Solution Methods

To our knowledge, all published work on the airline CAP follows the idea of generate-and-optimize or column generation, see Gamache and Soumis [17], Day and Ryan [14], and Ryan [28] for examples. The problem is divided into a subproblem where a subset of all legal rosters is generated, and a master problem where some of these rosters, one for each crew member, are selected. The master problem is a linear integer program whereas the structure of the subproblem is dependent on the rules, regulations, and objectives of the underlying CAP. Usually, one iterates between the master problem and the subproblem, thus gradually improving the solution quality.

[1] We will use *activity* and *task* synonymously.

The generation of legal rosters in the subproblem is either done by partial enumeration based on propagation and pruning techniques [23] or by solving a resource constrained shortest path problem where the constraints ensure that only legal rosters are generated. The objective function measures the reduced costs of the roster with respect to the solution of the continuous relaxation of the master problem defined on the previously generated rosters [17]. The latter approach is known as resource constrained shortest path column generation. In this approach, the subproblem can be solved optimally and one can prove that it is possible to obtain the optimal solution to the entire (relaxed) problem without explicit enumeration of all possible rosters.

Resource constrained shortest path column generation is a very powerful technique but it is only useful if rules, regulations and objectives satisfy certain properties (see [15] for an overview), whereas propagation and pruning techniques do not depend on particular assumptions on the problem.

Several alternative approaches have been proposed for solving the railway CAP, which differs slightly from the airline crew assignment, but also has to model rules and regulations. Caprara et al. [7] gave an algorithm that uses information from a Lagrangean lower bound based on the solution of an assignment problem to guide a heuristic. Their algorithm won the first prize in a competition organized by the Italian Railway Company. For the Lisbon Underground, Cavique et al. [10] used a tabu search procedure working on the entire solution. The contractual and operational rules in this case consist mainly of time limits for minimum/maximum work time and for meal breaks.

For the approach presented here, the work of Caprara et al. [8] is of certain interest. Their approach is mainly based on the constraint logic programming (CLP) paradigm enhanced with a lower bounding procedure taken from the operations research field. This is the natural choice since good and fast bounding procedures were available from [7]. The efficiency of the approach is supposed to be due to the existence of these bounding procedures for the case considered. Thus, a generalization to the generic airline CAP considered in this paper is not straightforward.

1.3 Integration of CP and OR Techniques

The integration of techniques from Constraint Programming (CP) and Operations Research (OR) has become an important field of research in recent years, see [20] for an overview. The integration of OR and CP techniques applied to airline CAP is investigated in the ESPRIT project Parrot (Parallel Crew Rostering) [27] where this work has been carried out. In principle there are two main ways of integrating CP and OR:

- Embed OR algorithms in a CP framework. [4, 6, 30] did this for a linear solver. The purpose of the OR algorithm is usually to provide bounds and possibly other information to guide the search. In certain cases, OR algorithms can also be exploited for domain reduction. A famous example are global resource constraints in scheduling that use the edge-finder [26].

- Embed CP algorithms in an OR framework. This is done in the context of a column generation algorithm. The master (OR) problem is a linear integer program and the difficult constraints are "hidden" in the column generating subproblem which is solved by CP.

This paper elaborates the latter approach. In our column generating subproblem we take advantage of the powerful modeling and solving abilities of CP for rules and regulations arising from a possibly non-linear world. For the master problem we use OR techniques.

An important result from the column generation (OR) theory says that new columns can improve the current solution to the continuous relaxation of the master problem, only if their reduced cost with respect to the current continuous solution is strictly negative. The major theoretical novelty of our work is to show how this result can be exploited efficiently by the CP algorithm applied to the subproblem. We do this by introducing a shortest path constraint and a negative reduced cost constraint allowing us to dynamically reduce domains in the subproblem using dual information from the master problem.

Compared to traditional approaches of generating columns, the CP framework combines a high expressiveness with the domain reduction capabilities of CP algorithms. In particular, the CP framework allows to encapsulate traditional generation techniques (e.g. sophisticated constrained shortest-path algorithms) inside a constraint and to exploit them for domain reduction. Constraints that cannot be treated by the shortest-path algorithm can now interact with it via domain reduction. We can also say that the additional constraints will cut illegal choices of the shortest-path algorithm as early as possible when searching for best columns. Thus, constraint programming based column generation can be seen as a way of enhancing or extending traditional approaches to column generation.

1.4 The Airline Test Case

For our experiments, we use data for several 4 week planning periods of Spring and Summer 1998 from a major European airline. The problem instances consider cockpit crew at one crew base and consist of around 400 crew members and around 1000 activities which have to be assigned.

For the crew assignment, the airline is interested in fair rosters which minimize overall costs. The costs are represented by a linear objective function on the rosters and the unassigned activities. The main costs of each roster depend on the total flight time (or block time) and costs on extra days off.

In the production system of that airline, around 90 single crew member rules are implemented. Representative subsets of rules and regulations have been chosen for our experiments. However, certain issues such as individual rule relaxations or restrictions are not considered.

The following lists seven rules which have been implemented for the test case presented in this paper. The formulations of some rules use the term *airline day* which is a 24 hours period not corresponding to a calendar day. It starts at time τ on one day and ends one minute before τ on the next day.

(R1) *Minimum Rest at Home Base:* A crew member gets at least an η_1-hour rest period at his/her home base between two activities.

(R2) *Minimum Rest for One Day Off:* If a crew member has one complete airline day off, i.e., the rest between two activities contains one airline day, the rest period must be at least η_2 hours.

(R3) *Minimum Rest for Two Days Off:* If a crew member has two complete airline days off, i.e., the rest between two activities contains two airline days, the rest period must be at least η_3 hours.

(R4) *Minimum Rest for Three Days Off:* If a crew member has three complete airline days off, i.e., the rest between two activities contains three airline days, the rest period must be at least η_4 hours.

(R5) *Latest Check-out for Two Days Off:* If an activity is followed by a rest period containing two airline days, the activity is not allowed to end in a *night period*, i.e., the period defined by an interval $[\tau_1, \tau_2]$.

(R6) *No Early Briefing After Five/Two:* After a 5-day working period followed by two airline days off, a crew member cannot be assigned an activity starting before time τ_3.

(R7) *No two trips on the same day:* Except for preassignments, it is not allowed to have two flight trips on the same day.

Additionally, crew members are required to have special activities like simulator training, medical checks, etc. during the planing period under consideration. These activities usually require interaction with resources that are not necessarily known in the planning system (e.g., slots for the simulator, medical personal, etc.). Therefore, in the present work they are not handled as special rules but simply by assigning these activities to a crew member in advance. We refer to these activities as *preassignments.*

Note, that due to the generic approach based on CP, the rule set can easily be extended and modified without significantly changing the algorithms described in the following sections.

1.5 Outline

The paper is organized as follows. The next two sections present the column generation approach for the CAP. In Sect. 2 we concentrate on the subproblem, especially we describe the new shortest path and the negative reduced cost constraint. The remaining components of the approach are introduced in Sect. 3. In Sect. 4 the new approach is applied to some airline crew assignment problems coming from a major European airline and numerical results are presented. Finally, we conclude in Sect. 5.

2 A CP Based Column Generation Approach

2.1 Column Generation

Column generation (also called Dantzig-Wolfe decomposition) is a well-known technique for handling linear programs (LPs) with a huge amount of variables:

$$\min c\,x, \quad \text{s.t. } A\,x = b \tag{1}$$

Its origins date back to the works of Dantzig and Wolfe [13] and Gilmore and Gomory [18]. The latter paper applies column generation to the classical cutting stock problem where the subproblem is a knapsack problem. More recent applications include specially structured integer programs such as the generalized assignment problem and time constrained vehicle routing, crew pairing, crew assignment and related problems. We refer to [15] for a survey.

Due to its size it is often impossible to solve the large system (1) directly. Column generation provides a way for obtaining the solution of this system indirectly. Therefore, a much smaller system $A'\,x = b$ is considered where A' contains only few columns of A. An optimal basic solution of the *restricted master problem* $A'\,x = b$ provides dual values λ_i of each constraint i of the linear system.

Now we pose the question: Which columns have to be added to A' yielding a linear system with the same solution value as the original problem (1)? Linear programming duality theory tells us that only columns with *negative reduced cost* can be candidates for entering the basis. This is the way the simplex algorithm chooses columns for its basis internally. But it can also be applied for the external generation of columns. Similar to the simplex algorithm, column generation can be stopped as soon as no further columns with negative reduced costs exist. Therefore, in the *subproblem* of column generation it is sufficient to search only for a column $\alpha := (\alpha_1, \ldots, \alpha_z)^T$ that obeys the negative reduced cost inequality

$$c_\alpha - \sum_{i=1}^{z} \lambda_i \cdot \alpha_i < 0 \tag{2}$$

where c_α denotes the cost coefficient for column α. α then is added to A'.

Theoretically, it may still be necessary to generate all possible columns before terminating the generation phase but in practice this rarely happens. Typically, only a small subset of all possible columns will be needed. Algorithm 1 describes this idea. There, we assume that function `solveSubproblem()` returns new columns $\{\alpha^{(1)}, \ldots, \alpha^{(k)}\}$ which are valid solutions for the subproblem, or an empty set, if no more solutions respecting Equ. (2) exist. An initial matrix A' is returned by `getInitialColumns()`. `solveLP()` solves the LP given by $A' x = b$ and returns the corresponding dual values λ. Function `addColumnsToMatrix()` extends A' by the columns generated.

Algorithm 1 Outline of the Column Generation

A' :=`getInitialColumns()`
repeat
  $\lambda :=$ `solveLP(`A'`)` *// get new duals*
  $\{\alpha^{(1)}, \ldots, \alpha^{(k)}\} :=$ `solveSubproblem(`λ`)` *// solve subproblem, respecting Equ. (2)*
  `addColumnsToMatrix(`$A', \alpha^{(1)}, \ldots, \alpha^{(k)}$`)`
until $(\{\alpha^{(1)}, \ldots, \alpha^{(k)}\} = \emptyset)$

For linear integer programs (IP) like the one presented in this work, however, column generation may not find the optimal solution as linear programming duality theory is not valid for IPs. In this case, two general approaches are possible.

The first approach is to ignore that fact: One solves the continuous relaxation of the problem first, and then applies branch-and-bound to obtain an integer solution. For a range of IP problems it is known that the remaining gap between LP and IP solution value is small enough to be neglected. We use a variant of this approach which will be described in Sect. 3.3.

Another possibility is to use branch-and-price where columns are generated in the nodes of a branch-and-bound tree and optimality can be proven. We refer to Barnhart et al. [2] for further information on this topic.

For the column generation approach used in this work, we decompose the CAP into a subproblem, where legal rosters that obey Equ. (2) are generated, and a set partitioning master problem ensuring that all activities are assigned exactly once.

2.2 The Subproblem

In this section, we present the subproblem of our column generation approach.

For each crew member we generate a set of rosters. A roster for crew member c is a set of activities Y that contains the chosen and possibly preassigned activities P of the crew member and that satisfies all single crew member rules. Each generated roster leads to the introduction of a new column in the master problem. For this purpose, we need the cost coefficient c_Y that is defined by this roster, as well as the coefficients $a_{i,Y}$ for each constraint of the master problem. These coefficients are uniquely defined by a roster and will be calculated when generating the roster. They also allow to check Equ. (2) and to generate only rosters with negative reduced costs.

The subproblem for each crew member consists of generating a set of rosters. For each roster we have to find a set Y of activities, a cost coefficient c_Y, and the coefficient $a_{i,Y}$ for the master constraint such that the following conditions are satisfied:

1. $P \subseteq Y \subseteq T$, where T is the set of all tasks that have to be assigned,
2. Y satisfies the single crew member rules, and
3. $c_Y - \sum_i \lambda_i \cdot a_{i,Y} < 0$.

In the following, we introduce a constraint programming approach in Sect. 2.3 and show how costs and reduced costs can be determined by finding a path in a graph (Sect. 2.4). We introduce and show how to encode the reduced cost constraint in this approach (Sect. 2.5). In order to obtain a powerful propagation, we additionally exploit the path-finding algorithm and encapsulate it in a path constraint (Sect. 2.6). Section 2.7 shows how the propagations of this constraint can be computed efficiently in linear time. Since the path finding approach alone does not take into account all airline rules, we introduce these additional constraints into the constraint model in Sect. 2.8. Finally, Sect. 2.9 presents some search heuristics for solving the subproblem.

2.3 Formulation as a CSP

In this section, we formulate the subproblem as a constraint satisfaction problem (CSP) (see [24, 25]) allowing us to express complex airline rules and regulations.

A CSP is specified by a set of variables and a set of constraints. Each variable x has an initial domain $D(x)$. This domain can be, for example, an interval of integers. A constraint consists of a tuple of variables $(x_1, \ldots, x_n)$ and an n-ary relation R which is a subset of $D(x_1) \times \cdots \times D(x_n)$. A solution of a CSP is an assignment of a value $v(x) \in D(x)$ to each variable x of the CSP such that all the constraints of the CSP are satisfied. A constraint with variables $(x_1, \ldots, x_n)$ and relation R is satisfied by a solution iff the tuple $(v(x_1), \ldots, v(x_n))$ is in R.

When defining a specific constraint with variables $(x_1, \ldots, x_n)$ we define the relation of this constraint as the set of all tuples $(\overline{x}_1, \ldots, \overline{x}_n)$ satisfying a given condition $C(\overline{x}_1, \ldots, \overline{x}_n)$. Thus, $\overline{x}$ is used to denote the value of x in the considered tuple.

CSPs are usually solved by applying domain reduction techniques. Each variable has a current domain $dom(x)$ that is initialized by $D(x)$. If x is an integer variable, we denote the smallest element of $dom(x)$ by $min(x)$ and the largest element by $max(x)$. We also call them the *lower* and *upper bounds* for x.

Each constraint has an associated domain reduction algorithm that tries to remove inconsistent values from the domains of its variables. An overall propagation algorithm propagates these domain changes among the constraints and iterates until no further constraint can reduce the domains [31]. In order to find a solution, domain reduction alone is not sufficient and a search mechanism is used additionally. Search decisions cause additional domain reductions by invoking the propagation algorithm. The alternative search decisions lead to a search tree that is usually explored by chronological backtracking[2]

Dead-ends (failures) in the search are obtained, if the current domain of a variable becomes empty, whereas a solution is found, if all variables are instantiated, i.e., they have exactly one element in their current domain.

[2]Non-chronological search methods such as best-first search and limited discrepancy search are also available.

For the problem of roster generation, we do not only use integer variables but also set variables. The value of a set variable is a set of integers selected from an initially given domain. The current domain of a set variable is defined by a lower and an upper bound, which are also called required set $req(Y)$ and possible set $pos(Y)$. The value of the set variable has to be a superset of $req(Y)$ and a subset of $pos(Y)$. Set variables replace an array of boolean variables. They lead to more compact constraint models and more efficient and powerful propagation algorithms.

For the roster generation problem of crew member c, we introduce a single set variable Y denoting the (unknown) set of tasks of c. The value of Y has to be a subset of the set of tasks T. Initially, the possible set $pos(Y)$ contains all tasks and the required set $req(Y)$ contains all preassigned tasks of the crew member. During roster generation, the required set contains all tasks that have already been assigned to the considered crew member, whereas the possible set contains activities that are still candidates to be assigned. Constraint propagation and search decisions will then remove tasks from the possible set or assign new tasks by adding them to the required set. If the required set is equal to the possible set, the set variable is bound and a solution has been found.

Most rules and regulations require the introduction of further constrained variables. For example, for ensuring a minimal rest time after a given number of days off we introduce a constrained integer variable d_i for each task t_i that represents the number of days off directly after this task.

We now give a small example illustrating effects of domain reduction during roster generation. Consider the set variable Y of a crew member c and suppose that its possible set $pos(Y)$ initially contains the four tasks t_1, t_2, t_3, t_4 satisfying following properties:

$$\begin{array}{rcccl} 0 & \leq & start(t_2) - end(t_1) & < & \eta_1 \\ 1day & \leq & start(t_3) - end(t_1) & < & \eta_2 \\ \eta_2 & \leq & start(t_4) - end(t_1) & & \end{array} \tag{3}$$

If crew member c has no preassigned tasks then the required set $req(Y)$ is empty initially. Now we start the roster generation procedure which explores different search decisions. For example, it assigns task t_1 to crew member c by adding it to $req(Y)$. This search decision leads to following domain reductions, which are all executed before another search decision is made.

1. Since t_1 is in $req(Y)$ and the rest between t_1 and t_2 is smaller than the minimal rest time η_1, the minimal rest time rule removes t_2 from $pos(Y)$.

2. The earliest task that can follow t_1 is now t_3. Since the rest time between these tasks is greater than 1 day the lower bound $min(d_1)$ of the variable d_1 representing the number of days off after t_1 is set to 1.

3. Since the rest time between t_1 and t_3 is smaller than the minimal rest time η_2 after 1 day off, the minimal rest time rule for 1 day off removes t_3 from $pos(Y)$.

Table 1 summarizes these domain reductions.

	Initial domain	Add t_1 to Y	Reduced domains
$req(Y)$	$\emptyset$	$\{t_1\}$	$\{t_1\}$
$pos(Y)$	$\{t_1, t_2, t_3, t_4\}$	$\{t_1, t_2, t_3, t_4\}$	$\{t_1, t_4\}$

Table 1: Domain reductions after an assignment

2.4 Shortest Path Problems

So far, we have not discussed how to calculate the coefficients c_Y and $a_{i,Y}$ (as introduced in Sect. 2.2). In CAPs, the cost c_Y of a roster is often determined by an additive cost function, and in cases where the cost is not completely additive, an additive function will usually be a good approximation. Furthermore, the dominating terms in Equ. (2) tend to be those associated with the λ coefficients, as there is a strong feasibility component in the problem. These terms are always additive. For each task t selected in Y, we obtain a cost c_t and the resulting cost of the roster is the sum of the c_t's of all selected tasks. The cost c_t does not only depend on task t, but also on the next task t'. For example, c_t can depend on the rest time after t which depends on the arrival time of t and the departure time of t'. In the following, we assume that roster costs are in fact defined by a sum of task costs c_t which depend on the selected tasks t and their successors t' (but not on any other tasks).

Due to this assumption, we can model the problem of finding a cheapest set of tasks by the problem of finding a shortest path in a weighted graph $G = (V, E)$. The nodes of the graph are the tasks plus an additional source s and an additional sink s'.

$$V = T \cup \{s, s'\} \tag{4}$$

If task t' has a start time $start(t')$ that is greater than the end time $end(t)$ of task t we introduce an edge (t, t'). Furthermore, we introduce an edge from the source s to any task t and an edge from any task t to the sink s':

$$E = \{(t, t') \in T \times T \mid end(t) \leq start(t')\} \cup \{(s, t),\, (t, s') \mid t \in T\} \tag{5}$$

Hence, the graph G is a directed acyclic graph and if we order the tasks in T by increasing start time we obtain a topological order $s, t_1, \ldots, t_n, s'$ on the nodes.

Each legal roster corresponds to a path from source to sink. The roster costs are obtained as the path costs if the edge weights are given as follows:

1. An edge from source s to node t is labeled with costs $\omega_{s,t} := 0$.
2. An edge from node t to node t' is labeled with the costs $\omega_{t,t'}$ that present the costs for executing t' directly after t.
3. An edge from node t to the sink node s' is labeled with the costs that t will have if it is the last task in the roster.

We can also represent the reduced costs of a roster in this way. We distinguish two kinds of constraints in the master problem. The first kind of constraint ensures that exactly one roster is selected for each crew member. We know a-priori that the coefficient a_i is 1, if constraint i concerns the considered crew member c and 0 otherwise. The second kind of constraint ensures that each task is covered by the required number of crew members. The coefficient a_i of the latter constraint will be 1, if the constraint concerns task t and if task t belongs to the selected path. Otherwise it will be 0. We can therefore simplify Equ. (2) as

$$-\lambda_c - \sum_{t \in Y} (\omega_{t,t'} - \lambda_t) < 0 \tag{6}$$

where λ_t is the dual value of master constraint for task t and λ_c is the dual value of master constraint for crew member c. Edge weights are then adapted correspondingly.

1. For edges from the source s to any task t we only add the negative dual of the crew member, i.e.,
$$c'_{s,t} = -\lambda_c. \tag{7}$$
2. Edges from node t to node t' (or to sink s') are labeled with the difference of the initial costs $\omega_{t,t'}$ and the dual value of task t:
$$c'_{t,t'} = \omega_{t,t'} - \lambda_t \tag{8}$$

These new costs $c'_{t,t'}$ reflect the reduced costs for performing activity t' after t. We call them also reduced costs of activity t and we distinguish them from the initial costs $c_{t,t'}$ of task t. Each roster with negative reduced costs then corresponds to a path from source to sink with negative costs in the graph with weights $c'_{t,t'}$. However, not every path from source to sink is a legal roster. We can already reduce a large number of illegal paths in a preprocessing step by removing edges. If task t' cannot follow task t since this violates some rule (e.g., on minimal rest time) we suppress the edge (t, t') in the graph. Rules that depend on more than two tasks cannot be treated in this way. For example rule (R6) requiring a day off after 5 days of work depends on more than two successive tasks. In the next sections, we show how constraint satisfaction techniques can be exploited to obtain dynamic reductions of the graph.

2.5 A Negative Reduced Cost Constraint

We show how to encode the negative reduced cost condition in the CSP-framework. The first idea is to use the Equ. (6) and to directly encode it by a sum-over-set constraint. For the sake of readability we do not specify the relation of the constraints, but just give the condition that the constraint is imposing on the value of Y.
$$\begin{aligned} -\lambda_c - \sum_{t \in \overline{Y}} \overline{c}'_t < 0 \\ \text{where} \quad \overline{c}'_t = \overline{c}_t - \lambda_t \end{aligned} \tag{9}$$
The initial costs c_t of task t and its 'reduced costs' c'_t are represented by constrained variables. The variable c_t can have a more complex definition involving next-in-set constraints (which will be explained in Sect. 2.8).

Although this is a straightforward way to express the negative reduced cost constraint, we have to analyze how effective it is. The negative reduced cost constraint is a very tight constraint. If it does not reduce the domains sufficiently early during the search, much search effort will be spent in subtrees of the search space which will not contain any roster of negative reduced cost.

We only have an upper bound on the reduced costs of a roster, namely 0. The sum-over-set constraint uses this upper bound as follows. First, it determines a necessary lower bound lb by summing up the lower bounds $min(c'_t)$ of the tasks. If lb is strictly greater than 0 an inconsistent search state is reached and backtracking occurs. Otherwise, if the bound lb is smaller than 0 the constraint checks for each element t in $pos(Y) \setminus req(Y)$ whether its selection would make the lower bound greater than 0. If so, the task t will be removed from the possible set.

This method is only efficient, if a good lower bound is computed. The lower bound is defined as follows:
$$lb := -\lambda_c - \sum_{t \in req(Y), min(c'_t) \geq 0} min(c'_t) + \sum_{t \in pos(Y), min(c'_t) < 0} min(c'_t) \tag{10}$$

Suppose that almost all tasks could have negative reduced costs c'_t. The sum-over-set constraint does not know any incompatibilities between tasks and assumes that all tasks in $pos(Y)$ could be selected. In this case, we obtain a very small negative lower bound, which probably will not lead to any domain reductions. Domain reductions will only occur deep in the search tree when the size of the possible set comes close to the size of the required set.

The graph introduced in Sect. 2.4 avoids a large number of incompatible tasks. If task t' cannot follow task t there is no edge from t to t'. A path of the graph never contains two successive tasks that are incompatible. As a consequence, a shortest path in the graph leads to a much tighter lower bound and to better propagation.

In the next section, we introduce a path constraint that implements this approach and that provides a significant improvement over the sum-over-set constraint. We nevertheless keep the sum-over-set constraint in order to be able to measure the improvements. In our experiments we use a simplified implementation of this sum-over-set constraint that avoids all calculations for propagations which will probably not be effective anyway. It just provides the basic pruning rule that compares the calculated lower bound lb with the given upper bound 0. Hence, no domain reduction is caused by this constraint. It thus provides an interesting basis for comparisons, because it is fast and does not lead to any computational overhead.

2.6 An Efficient Path Constraint

In this section, we introduce a path constraint for acyclic graphs that provides a powerful propagation for the negative reduced cost condition.

The path constraint is defined for a directed acyclic graph $G = (V, E)$, the edge costs $c_{i,j}$, a set variable Y, and a variable z. The graph G has the same form as in Sect. 2.4 except that we use numbers for representing the nodes. If the tasks are ordered by increasing start time as follows $t_1, \dots, t_n$ we replace t_i by i, s by 0, and s' by $n+1$. Since there is no edge from i to j if $j < i$, the node numbers represent a topological order.

To simplify the notation we define $t_0 := s$ and $t_{n+1} := s'$. The path constraint then has the variables Y and z and is defined by two conditions:

1. Y represents a path in the graph from source s to sink s', i.e., $\overline{Y} \subseteq T$ and

$$\{(i,j) \mid t_i \in \overline{Y} \cup \{s\},\ t_j = next(t_i, Y)\} \subseteq E \tag{11}$$

2. z is the sum of the edge costs

$$\overline{z} = \sum_{\substack{t_i \in \overline{Y} \cup \{s\}, \\ t_j = next(t_i, Y)}} c_{i,j} \tag{12}$$

where $next(t, Y)$ denotes the successor of t on the path Y. Since we have a topological order on G, this successor is uniquely defined as

$$next(t, Y) = min\{t' \in \overline{Y} \cup \{s'\} \mid t' > t\} \tag{13}$$

$$\text{where} \quad t_j > t_i \quad \text{iff} \quad j > i.$$

In the following, we introduce propagations of the path constraint that are important for roster generation. In this case, we just have the upper bound 0 on the path costs z. The propagations require the calculations of paths from source to sink that contain all required tasks and only possible tasks (i.e., the set of nodes Y of such a path has to satisfy $req(Y) \subseteq Y \subseteq pos(Y)$). We call such a path *admissible*. The propagation rules are:

1. If the bound $min(z)$ is smaller than the costs lb of the shortest admissible path (or $+\infty$ if there is no admissible path) we replace it by lb. If the new bound is strictly greater than the upper bound $max(z)$, a failure is obtained as a consequence.

2. If the costs of the shortest admissible path through node i for $i \in pos(Y)$ are strictly greater than the upper bound $max(z)$ we can remove i from $pos(Y)$.

In the next section, we show how shortest admissible paths can be computed efficiently. Furthermore, we discuss how to maintain these paths when incremental updates occur.

2.7 Implementation

To compute a lower bound on the reduced costs, we just have to solve the *single source shortest path problem* (SSSP) starting with node s. We denote the single source shortest path distance from s to s' by $y^s_{s'}$ and call it the *SSSP-distance*. The SSSP-distance is the desired lower bound on the reduced costs of all legal rosters, as each of them can be identified with a (s, s')-path.

To solve the SSSP, we apply an algorithm that makes use of the acyclic structure of the graph. It visits nodes in topological order and thus runs in linear time $O(|V| + |E|)$. Furthermore, it does not require that edge costs are positive (see [12] for details).

However, we have to ensure that the algorithm determines only nodes which are subsets of $pos(Y)$ and supersets of $req(Y)$. For this purpose, we consider only nodes in $pos(Y)$ and we ignore all edges (i, j) that go around a node of $req(Y)$. That means, if there is a $k \in req(Y)$ s.t. $i < k < j$ we do not consider (i, j). This can be done efficiently by determining the smallest element $k \in req(Y)$ that is strictly greater than i and ignoring all edges (i, j), $\forall j > k$.

During the search for a legal roster, the domain of Y changes frequently, which means that many and often similar SSSPs have to be solved. We therefore developed an incremental version of the algorithm that computes the differences in the domains of the current and the last iteration. The required set may have grown and the possible set may have shrinked. If a new node became required, we need to restart the SSSP-algorithm with this node and can stop when we first run over an already formerly required node; its difference in distance then applies to all following nodes as well. The algorithm follows the support idea in the style of AC-6 [5]. If a node i' left the possible set, we check if any adjacent node i is affected by that change. We call i' the *support node* for i.

If the node i is affected, its support may be replaceable by another node without a change in the distance y^s_i to the start node s. If this is not the case, we need to do the distance update and mark the successors of node i for a continuing update on all affected nodes. Notice, that the updates for all nodes can be done in one pass.

Of course, if the lower bound becomes greater than zero, the search for a legal roster with negative reduced costs fails and we backtrack. If not, we can use the distance information just computed to reduce the domain of Y further.

To eliminate activities from $pos(Y)$, we must determine the cost of a shortest admissible path going through node $i \in pos(Y)$. For directed acyclic graphs, this cost is simply the sum of the costs y^s_i of the shortest path from the source s to node i and the costs $y^{s'}_i$ of the shortest path from i to the sink s'. The latter can be computed with the same algorithm by just inverting all the edges and by applying it to the sink s'. If $y^s_i + y^{s'}_i$ is non-negative, we remove i from the possible set. These ideas are summarized in Alg. 2. To have a compact representation we omit the technical details of the incremental variant.

Algorithm 2 (Non-Incremental) Shortest Path Constraint

```
for all $i \in T$ do
  $y_i^s := \infty$; $y_i^{s'} := \infty$;
$y_s^s := 0$; $y_{s'}^{s} := 0$ // Init source and sink
$k := 0$;
// determining shortest path from source s to all nodes
for all $i \in V$ taken in increasing topological order do
  if $k \le i$ then
    $k := \min\{l \in req(Y) \mid l > i\}$; // get next required node in the top. ordering
  for all $j \in pos(Y)$ s.t. $(i,j) \in E$ and $j \le k$ do
    if $y_j^s > y_i^s + c_{i,j}$ then
      $y_j^s := y_i^s + c_{i,j}$
// determining reverse shortest path from sink s' to all nodes
$k := 0$;
for all $i \in V$ taken in decreasing topological order do
  if $k \ge i$ then
    $k := \max\{l \in req(Y) \mid l < i\}$; // get next required node in the top. ordering
  for all $j \in pos(Y)$ s.t. $(j,i) \in E$ and $j \ge k$ do
    if $y_j^{s'} > y_i^{s'} + c_{j,i}$ then
      $y_j^{s'} := y_i^{s'} + c_{j,i}$
if $y_{s'}^s > min(z)$ then
  $min(z) := y_{s'}^s$ // propagate new lower bound on costs
for all $i \in pos(Y)$ do
  if $y_i^s + y_i^{s'} > max(z)$ then
    $pos(Y) := pos(Y) \setminus \{i\}$ // exclude nodes that are too expensive
```

2.8 Single Crew Member Rules

The rules and regulations coming from legislation and contractual agreements are formulated by constraints over set variables, such as cardinality constraints, sum-over-set constraints, next- and previous-in-set constraints, or set-of constraints. Most of them are provided by the constraint library ILOG SOLVER, others have been implemented as extensions of SOLVER. Given the flight time f_t (block hours) of each task t and a maximal flight time F, we can express a maximal flight time rule by a sum-over-set constraint

$$\sum_{t \in \overline{Y}} f_t \le F. \tag{14}$$

In order to express a minimal rest time rule between two successive tasks, we need the arrival time $end(t)$ of a task t and the departure time $start(t')$ of the next task t'. Given a minimal rest time η between two tasks, the minimal rest time rule can then be expressed by the next-in-set constraint (13) as follows:

$$start(next(t, Y)) - end(t) \ge \eta \tag{15}$$

where $next(t, Y)$ is defined as in Equ. (13).

Regulations on days off, maximal duration of working periods, and so on, can be expressed as well, but usually lead to more complex constraint models. In order to facilitate this modeling task, a specific ROSTER LIBRARY has been developed in the scope of the PARROT project [27]. Complex crew regulations that have been formulated by expressions of the ROSTER LIBRARY are automatically translated into compact constraint models of SOLVER.

In real world CAPs the rules and regulations will vary from crew member to crew member. Quite often individuals only differ with respect to parameters (e.g. reduced flying time). This doesn't affect the number of rules or the structure of these. It may also be, that there has to be some kind of compatibility between attributes (e.g. qualifications) of the crew member and attributes (requirements) of the tasks. This is also modeled with generic rules, where the results clearly depend on the crew member in question. A common rule requires that the aircraft type flown must be in the set of aircraft types, for which the crew member is qualified. We may have cases where different crew members work under completely different agreements. This increase the number of rules, but not necessarily the number of constraints, as all rules will not apply to all crew members.

Since the resulting models become quite complex, we illustrate the possible propagations just for two rules, namely (R1) and (R2). For (R1) (Minimum Rest at Home Base) we get a constraint model that excludes all those activities from the possible set of Y that cannot follow legally any already required activity. Rule (R2) (Minimum Rest for One Day Off) is modeled such that any possible activity that follows a required one with one airline day in between is excluded from further search, if the rest time is less than η_2. The corresponding propagation rules are given in Alg. 3.

Algorithm 3 Propagation Algorithm for Rules (R1) and (R2)

```
for all t ∈ req(Y) do
  for all t' ∈ pos(Y) \ req(Y) do
    if (start(t') − end(t) < η1) then
      pos(Y) := pos(Y) \ {t'} // Rule R1
```

```
for all t ∈ req(Y) do
  for all t' ∈ pos(Y) \ req(Y) do
    if (one_airline_day(t,t') AND (start(t') − end(t) < η2)) then
      pos(Y) := pos(Y) \ {t'} // Rule R2
```

2.9 Search Heuristics

The shortest path constraint and the negative reduced cost constraint help to reduce the search effort in pruning parts of the search tree without using the special structure of the underlying problem. However, for moderate to large size real-world problems this pruning alone is not sufficient for finding promising rosters. Heuristics designed for the special structure of the CAP are needed to prune further and/or guide the search to promising regions of the search space. The search space is explored by Alg. 4.

Algorithm 4 A Non-Deterministic Algorithm Describing the Search Tree

```
while (pos(Y) ≠ req(Y)) do
  1: select a task t ∈ pos(Y) \ req(Y) s.t. t is on a shortest admissible path
  2: chose (req(Y) := req(Y) ∪ {t}) OR (pos(Y) := pos(Y) \ {t})
  3: apply constraint propagation algorithm
```

The task selection (Step 1 in Alg. 4) should take care of first selecting *the best* activity from the possible set. It is often possible to sort activities during preprocessing (*static ordering*). In case the order of activities depends on the search history, the ordering can also be carried out during the search (*dynamic ordering*).

As static ordering method one can choose the first possible successor of the last activity that is already required. This first activity first approach is believed to be good at generating productive rosters with only little idle time between activities. In general, these rosters have a low cost, due to the fact that airlines typically prefer such rosters to those with much unproductive time between two activities. This static ordering is also used by Ryan [28].

We get a dynamic ordering when using the shortest path constraint and choosing the pairing on the path that contributes the lowest value to the path costs. The *Lowest Reduced First (LRF)* method selects a task with the lowest reduced cost and proves to be superior in our experiments. The advantage of LRF is that the costs contributed by the pairing to the resulting costs of the roster are also taken into account.

For the branching order (Step 2 in Alg. 4) we try to add the selected task to the required set of Y in the first branch and to remove it from the possible set of Y on the second branch.

3 The Entire Approach

3.1 The Master Problem

To combine the rosters generated by solving the subproblem to one entire work plan, the master problem has to be solved. It consists of minimizing the total cost (the sum of all chosen rosters) by choosing exactly one line of work for each crew member, such that all pairings are assigned exactly once. This problem can be stated as binary IP problem, or – to be more precise – as a *set partitioning problem (SPP)*. However, we are also interested in the *set covering problem (SCP)* as we use it as a relaxation of the SPP. These two problems are defined as

$$\text{SPP:} \qquad \min c\,x, \;\; \text{s.t. } A\,x = 1, \quad x \text{ binary} \tag{16}$$

$$\text{SCP:} \qquad \min c\,x, \;\; \text{s.t. } A\,x \geq 1, \quad x \text{ binary} \tag{17}$$

where A is a 0-1 matrix, and c is the cost vector for the columns. Both problems are well studied and good algorithms solving (16), (17) can be found in the literature (see e.g., [9, 19]).

A legal roster Y for a crew member is translated into a column of matrix A by inserting 1's into the row corresponding to the crew member and into all rows that correspond to activities used in Y. Especially in the beginning, however, it is unlikely and hard to guarantee that the columns generated so far will fit together exactly. Having n crew members and m activities, we extend A by unit-vectors e_i, $i = 1, \ldots, n + m$, ensuring that the resulting set partitioning problem always has a feasible solution. The so-called *dummy rosters* can be interpreted as crew members without work ($i = 1, \ldots, n$), and uncovered activities ($i = n+1, \ldots, n+m$), respectively. To find meaningful solutions, we have to penalize uncovered pairings by assigning high costs to the columns representing them.

Obviously, solutions consisting of many dummy rosters are not of interest. Therefore we relax the last n constraints to SCP constraints, i.e., every activity must be covered *at least* once. Due to the high costs of the dummy rosters, a solution will contain few or even no dummy rosters. But in the solution some of the chosen columns might collide in the sense that there are activities that are assigned more than once. Then, a solutions respecting SPP constraints is heuristically obtained by adding new legal subrosters extracted from colliding ones.

However, in some cases it is not possible to simply fix collisions without generating new rosters in the subproblem. E.g., a rule limiting the minimum working time for a roster might not allow to build subrosters of a colliding roster. In such a case, the solution of the SPP will contain dummy rosters and the subproblem has to generate different rosters from scratch using new dual information.

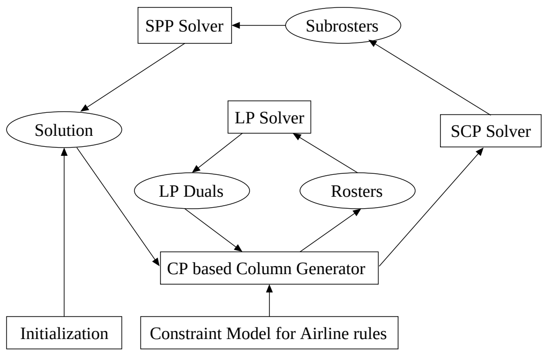

Figure 1: The entire approach: The inner loop generates columns using dual information, the outer loop solves the master problem.

3.2 Multiple Crew Member Rules

In real world applications of crew assignment, there is also a class of multiple crew member rules that involve properties of *several* rosters. E.g., for cabin crew it might be necessary to have at least n_1 people speaking a certain language on a certain flight. Another typical example is to have at most n_2 unexperienced persons in the cockpit. As the subproblem only handles single rosters, these rules have to be handled in the master problem.

A simple way of integrating linear multiple crew member constraints (as the ones sketched above) is to extend the SPP by additional constraints. These constraints are usually expressed by a weighted sum over some attributes of the crew members. The only complication arising then is to compute the reduced costs when generating rosters in the subproblem. To do this, the column generator just needs to have knowledge about the coefficients of the multiple crew member constraint that will arise when adding an activity to a roster. In essence Equ. (10) is adapted by adding dual values of all those multiple crew member rules in which the current crew member provides a nonzero attribute.

Therefore, multiple crew member rules do not change the behavior of the negative reduced cost constraint in principal. Thus to keep the setting simple we will not use multiple crew member rules in the experiments (Sect. 4).

3.3 Overview of the Approach

With the different components discussed before, we are now able to describe their interaction. First, we set up the master problem and add dummy rosters (i.e., crew members without work and unassigned activities) to ensure the existence of a solution. We additionally append a certain number of initial rosters for each crew member. Then we initialize the column generator and define the search strategies to be used.

Entering the (outer) loop of *master iterations* (see Fig. 1), we solve the current SPP-IP and get a first current solution and new dual values of the LP-relaxation. The column generator then defines the next subproblem to be solved by picking a crew member and maybe additionally applying other search limitations as, e.g., fixing some assignments according to a time window focus and the current SPP solution. We start generating a specified number of rosters, then add the corresponding columns to the master problem, solve its continuous relaxation, and update the current dual values. This inner loop corresponds to Alg. 1.

After rosters have been generated for all crew members, we solve the set covering

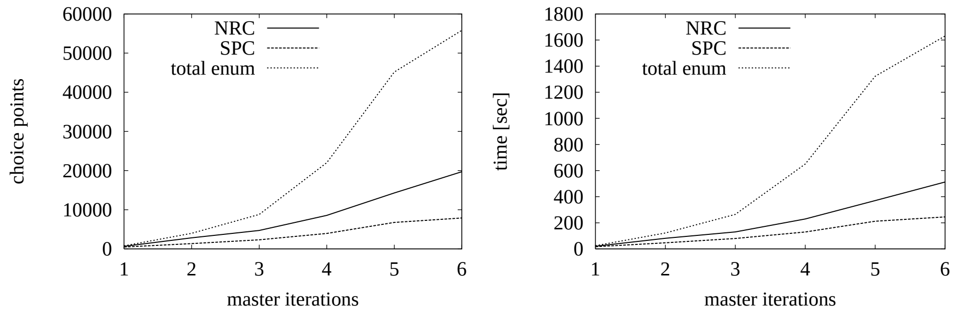

Figure 2: Number of choice points versus master iteration (left), and running time versus master iteration (right) for SPC, NRC, and total enumeration. The tests were run with a data instance of type 10-00-20 that was solved to optimality.

relaxation of the SPP master problem and generate subrosters of the rosters used in the SCP optimal solution to resolve collisions in the SPP. Afterwards, the SPP itself is being solved and a new loop begins.

This process is either interrupted after a given time limit has been exceeded or when no more rosters have been generated that yield to an improvement of the LP-relaxation in any of the subproblems.

4 Numerical Results

In this section, we show that constraint programming based column generation is able to solve non-trivial crew assignment problems. In particular, we demonstrate the effect obtained by the propagation of the path constraint.

As mentioned before, the problem instances used for the experiments stem from a major European airline. The rules, regulations, and objective function have directly been abstracted from the real-world case and preserve the essential characteristics of this case. The data sets are sufficiently large to measure the effects of constraint propagation, but they are small enough to run experiments in a reasonable time frame.

To characterize an instance, we specify the number of crew members, the number of preassigned activities, and the number of activities to be assigned. For example, an instance of type 67-165-280 consists of 67 crew members, 165 preassignments, and 280 tasks. All experiments were run on a SUN Ultra 4 with 296 MHz CPU and 1024 MB main memory. For the constraint model of the CAP, the Roster Library [27] based on Ilog Solver 4.4 [21] was used. The LP and IP problems were solved with Ilog Planner 3.3 [22].

It is important to note that the size of a problem instance is not only determined by the number of crew members and tasks, but also by the number of subtasks (e.g. flight legs and ground duties) and the number of attributes per tasks. In the example above, the 280 tasks consist of 1422 sub-tasks. The given rules and regulations are formulated with the help of 66 attributes per task, 32 per sub-task, and 7 per crew member. A part of these attributes is translated into constrained variables.

In the experiment for Fig. 2 we show the effects of negative reduced cost constraint (NRC) and shortest path constraint (SPC). We compare both results with a total enumeration, where neither NRC nor SPC is used to reduce the search space. The left picture shows the reduction of choice points. In the end SPC used less than half the number of choice points than the NRC. This gain is not consumed by a significant increase in computation time per choice point. As shown in the left

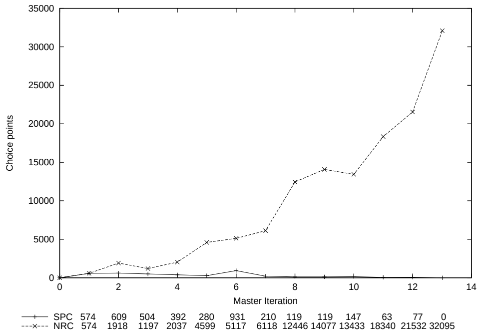

Figure 3: Number of choice points versus master iteration using SPC, NRC with a data set of type 7-0-30.

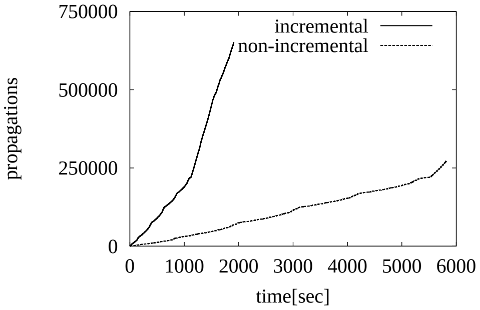

Figure 4: The picture shows time versus the number of calls of the propagation routine using the incremental and the non-incremental implementation of the shortest path constraint. Both versions were stopped after 10 000 seconds total CPU time. The experiment was run with a data instance of type 10-00-70.

figure the decrease in running time is quite similar to the decrease in the number of choice points. As expected, total enumeration is not competitive at all.

To demonstrate the superiority of the shortest path constraint against the negative reduced cost constraint in more detail we run a test on a small instance where in each master iteration the number of choice points was noted. Figure 3 again shows that the shortest path constraint uses much less choice points than the negative reduced costs constraint. Furthermore, in the last iteration the shortest path constraint does not find any columns with negative reduced costs anymore, thus proving optimality for the continuous relaxation of the master problem. The negative reduced cost constraint, however, still visits an increasing number of choice points per iteration.

One reason for the efficiency of the shortest path constraint and the reason why there is almost no gap between the reduction of choice points and the reduction in time is the use of the incremental version as mentioned in Sect. 2.7. In Fig. 4 we compare a non-incremental version of the shortest path constraint with an incremental one. For a fixed time of 10 000 sec. for the entire optimization the faster incremental version uses only 2 000 seconds for the propagation, whereas for the non-incremental version almost 60% of total calculation time goes into that part of the algorithm. Thus, the incremental version allows to perform nearly 3 times as many propagations as the non-incremental version and hence helps to improve solution quality.

Figure 5 shows a time versus quality comparison of NRC and SPC. After a first

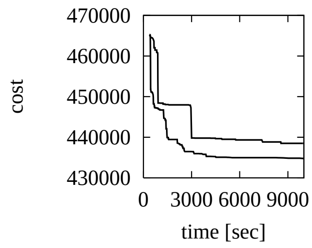

Figure 5: Time versus quality on a data instance of type 67-165-280. The picture shows a comparison of NRC (upper curve) and SPC (lower curve).

big drop in the objective, the NRC falls into huge search trees that only consist of rosters with non-negative reduced costs. The SPC can prune those search trees much earlier and therefore continuously reduces the objective without stalling.

The numerical results clearly prove the potential of SPC and that the overhead created in the subproblem pays off when comparing it to NRC. However, our experiments also showed that there is room for improving the generation subproblem, using various techniques to limit the search. The present work focused on evaluating a generic framework applied to the airline crew assignment problem. Ongoing work by the authors investigates how the approach can be improved and tailored to the problem to meet performance requirements for a real world application.

5 Conclusions

We have introduced a CP-based column generation approach for the airline CAP. For the case of European airlines with complex rules and regulations common approaches using constrained shortest path algorithms to solve the subproblem in a column generation framework are limited. We therefore formulated the subproblem as a constraint satisfaction problem. Tests with real data from a major European airline showed that the development of a new shortest path constraint combining methods from CP and OR yields a significant decrease in the number of choice points during the generation. An incremental update implementation of this constraint reduces the computational effort per choice point, such that overall computation times are reduced significantly as well. Branching variable selections based on shortest path information further improve the performance.

The presented approach is an example of a successful integration of CP and column generation. We believe that this approach will prove useful for a number of important optimization problems which are currently solved using only OR or only CP methods. There are still refinements and improvements to be done, but this work demonstrates the applicability and efficiency of the approach.

References

[1] E. Andersson, E. Housos, N. Kohl, and D. Wedelin. Crew pairing optimization. in G. Yu (editor): Operations Research in the Airline Industry, pp. 228–258, *International Series in Operations Research and Management Science*, Vol. 9, Kluwer Academic Publishers, 1998.

[2] C. Barnhart, E.L. Johnson, G.L. Nemhauser, M.W.P. Savelsbergh, and P.H. Vance. Branch-and-price: Column generation for solving huge integer programs. *Operations Research*, 46(3):316–329, 1998.

[3] C. Barnhart and R.G. Shenoi. An approximate model and solution approach for the long-haul crew pairing problem. *Transportation Science*, 32(3):221–231, 1998.

[4] H. Beringer and B. De Backer. Combinatorial problem solving in constraint logic programming with cooperative solvers. In C. Beierle and L. Plumer (editors), *Logic Programming: Formal Methods and Practical Applications*, pp. 245–272. Elsevier, 1995.

[5] C. Bessière. Arc-consistency and arc-consistency again. *Artificial Intelligence*, 65:179–190, 1994.

[6] A. Bockmayr and T. Kasper. Branch and infer: A unifying framework for integer and finite domain constraint programming. *INFORMS Journal on Computing*, 10(3):287–300, 1998.

[7] A. Caprara, P. Toth, D. Vigo, and M. Fischetti. Modeling and solving the crew rostering problem. *Operations Research*, 46(6):820–830, 1998.

[8] A. Caprara, F. Focacci, E. Lamma, P. Mello, M. Milano, P. Toth, and D. Vigo. Integrating constraint logic programming and operations research techniques for the crew rostering problem. *Software – Practice and Experience*, 28(1): 49–76, 1998.

[9] A. Caprara, M. Fischetti, and P. Toth. A heuristic algorithm for the set covering problem. *Integer Programming and Combinatorial Optimization. 5th International IPCO Conference Proceedings*, pp. 1–15, Springer, 1996.

[10] L. Cavique, C. Rego, and I. Themido. Subgraph ejection chains and tabu search for the crew scheduling problem. *Journal of the Operational Research Society*, 50:608–616, 1999.

[11] H.D. Chu, E. Gelman, and E.L. Johnson. Solving large scale crew scheduling problems. *European Journal of Operational Research*, 97:260–268, 1997

[12] T.H. Cormen, C.E. Leierson, and R.L. Riverste. *Introduction to Algorithms*, McGraw-Hill, 1990.

[13] G.B. Dantzig and P. Wolfe. The decomposition algorithm for linear programs. *Econometrica*, 29(4):767–778, 1961.

[14] P.R. Day and D.M. Ryan. Flight attendant rostering for short-haul airline operations. *Operations Research*, 45(5):649–661, 1997.

[15] J. Desrosiers, Y. Dumas, M.M. Solomon, and F. Soumis. Time constrained routing and scheduling. In Ball, Magnanti, Monma, Nemhauser (editors): Network Routing. *Handbooks in Operations Research and Management Science*, Vol. 8, pp. 35–139, North-Holland, 1995.

[16] G. Desaulniers, J. Desrosiers, Y. Dumas, S. Marc, B. Rioux, M.M. Solomon, and F. Soumis. Crew pairing at Air France. *European Journal of Operational Research*, 97:245–259, 1997.

[17] M. Gamache, F. Soumis, D. Villeneuve, J. Desrosiers, and E. Gélinas. The preferential bidding system at Air Canada. *Transportation Science*, 32(3):246–255, 1998.

[18] P.C. Gilmore and R.E. Gomory. A linear programming approach to the cutting stock problem. *Operations Research*, 9:849–859, 1961.

[19] K.L. Hoffman and M. Padberg. Solving airline crew scheduling problems by branch-and-cut. *Management Science*, 39(6):657–682, 1993.

[20] J. Hooker. Unifying optimization and constraint satisfaction. *Invited talk at IJCAI '99*. Slides available at `http://ba.gsia.cmu.edu/jnh/ijcai.ppt`

[21] Ilog Solver 4.4. Reference manual and user manual. ILOG, 1999.

[22] Ilog Planner 3.3. Reference manual and user manual. ILOG, 1999.

[23] N. Kohl and S.E. Karisch. Airline crew assignment: modeling and optimization. Carmen Report, 1999. In preparation.

[24] A.K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8(1):99–118, 1977.

[25] U. Montanari. Networks of constraints: fundamental properties and applications. *Information Science*, 7(2):95–132, 1974.

[26] W.P.M. Nuijten and E.H.L. Aarts. A computational study of constraint satisfaction for multiple capacitated job shop scheduling. *European Journal of Operational Research*, 90(2):269–284, 1996.

[27] Parrot. Executive Summary. ESPRIT 24 960, 1997.

[28] D.M. Ryan. The solution of massive generalized set partitioning problems in aircrew rostering. *Journal of the Operational Research Society*, 43(5):459–467, 1992.

[29] R.A. Rushmeier, K.L. Hoffman, and M. Padberg. Recent advances in exact optimization of airline scheduling problems. Technical Report, George Mason University, 1995.

[30] R. Rodosek, M. Wallace, and M.T. Haijan. A new approach to integrating mixed integer programming and constraint logic programming. *Annals of Operations Research*, 86:63–87, 1999.

[31] P. Van Hentenryck, Y. Deville, and C.M. Teng. A generic arc-consistency algorithm and its specializations. *Artificial Intelligence*, 57:291–321, 1992.

[32] G. Yu (editor). Operations Research in the Airline Industry. *International Series in Operations Research and Management Science*, Vol. 9, Kluwer Academic Publishers, 1998.