

A Framework for Constraint Programming Based Column Generation[★]

Ulrich Junker¹, Stefan E. Karisch², Niklas Kohl², Bo Vaaben³, Torsten Fahle⁴,
and Meinolf Sellmann⁴

¹ ILOG S.A., 1681, route des Dolines, F-06560 Valbonne, France,
`junker@ilog.fr`

² Carmen Systems AB, Odinsgatan 9, S-41103 Gothenburg, Sweden,
`stefank@carmen.se` and `niklas@carmen.se`

³ Technical University of Denmark, Department of Mathematical Modelling,
Building 321, DK-2800 Lyngby, Denmark,
`bo.vaaben@sas.dk`

⁴ University of Paderborn, Department of Mathematics and Computer Science,
Fürstenallee 11, D-33102 Paderborn, Germany,
`tef@uni-paderborn.de` and `sello@uni-paderborn.de`

Abstract. Column generation is a state-of-the-art method for optimally solving difficult large-scale optimization problems such as airline crew assignment. We show how to apply column generation even if those problems have complex constraints that are beyond the scope of pure OR methods. We achieve this by formulating the subproblem as a constraint satisfaction problem (CSP). We also show how to efficiently treat the special case of shortest path problems by introducing an efficient path constraint that exploits dual values from the master problem to exclude nodes that will not lead to paths with negative reduced costs. We demonstrate that this propagation significantly reduces the time needed to solve crew assignment problems.

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

1 Introduction

The column generation method, also known as Dantzig-Wolfe decomposition, is a powerful method for solving large-scale linear and integer programming problems. Its origins date back to the works of Dantzig and Wolfe [6] and Gilmore and Gomory [9].

[★] The production of this paper was supported by the PARROT project, partially funded by the ESPRIT programme of the Commission of the European Union as project number 24 960. The partners in the project are ILOG (F), Lufthansa Systems (D), Carmen Systems (S), Olympic Airways (GR), University of Paderborn (D), University of Athens (GR). This paper reflects the opinions of the authors and not necessarily those of the consortium.

Column generation is a method to avoid considering all variables of a problem explicitly. Take as an example a linear program with an extremely large number of variables. We could solve this problem by only considering a small subset X' of the set of variables X . The resulting problem is usually denoted the *master problem*. Once it is solved, we pose the question: “Are there any variables in $X \setminus X'$ which can be used to improve the solution?”. Duality theory provides a necessary condition that a variable with negative reduced cost is the right choice and the simplex algorithm tries to find such a variable by explicitly calculating the reduced cost of all variables. The column generation idea is to find the variables with negative reduced costs without explicitly enumerating all variables. In the case of a general linear program this is not possible, but for many kinds of problems it is possible, as we shall see. The search for variables with negative reduced costs is performed in the so called *subproblem*. Theoretically, this may still require the generation of all variables. However, in practice such behavior is rare.

In the discussion above the master problem was assumed to be a linear program. In many applications, including the one described in this work, the master problem is a mixed integer linear program (MIP). This is a complication because the linear programming duality theory is not valid for MIPs, and there is no easy way to characterize a variable which may improve the current solution. In practice one solves the continuous relaxation of the problem first, and then applies branch-and-bound to obtain an integer solution. We will not discuss this issue further and refer to [7] for a more detailed discussion. The subproblem consists of finding the variables which will have negative reduced costs in the current master problem. Usually the subproblem is not a linear program, but rather a specially structured integer program. However, this does not constitute a problem as it can be solved reasonably fast.

Column generation has been applied to a large number of problems. The first application consisted of specially structured linear programs. An other classical application is the so called “cutting stock problem” [9] 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 [7].

In this work we consider the integration of column generation and constraint programming (CP). The problem may be formulated as a linear integer program with an extremely large number of variables. Again, since one cannot consider all these explicitly, one wants to generate the variables as needed. Often, the subproblem contains a large number of non-linear constraints and it is therefore not so well suited for the traditional operations research (OR) algorithms. Instead, we propose to apply a CP approach to solve the subproblem. The column generation theory tells us, that a useful variable must have a negative reduced cost in the current master problem. This can obviously be used to introduce a negative reduced costs constraint in the CP model.

The main contribution of our work is the bridging of CP and OR by introducing a general framework for column generation based on CP. Moreover, we

show how to efficiently implement an important special case of this framework, i.e. we establish the valid propagation rules and show how the propagation can be performed efficiently. The special case of the framework is then applied successfully to the airline crew assignment problem, a difficult large scale resource allocation problem with a huge number of complex constraints. The integration of OR and CP techniques applied to airline crew assignment is investigated in the ESPRIT project PARROT (Parallel Crew Rostering) [14] where also this work has been carried out.

The usefulness of column generation compared to a direct CP/LP approach has, for example, been demonstrated in the ESPRIT project CHIC II for a large-scale resource allocation problem with maintenance scheduling [12]. We go a step further and address problems where traditional column generation cannot be applied.

The organization of this paper is as follows. Section 2 presents the case study of airline crew assignment. The framework for CP based column generation is introduced in Sect. 3 and efficient propagation methods for an important special case of the framework are described in Sect. 4. In Sect. 5 the new generation approach is applied to the airline crew assignment problem and numerical results based on real data are presented which indicate the effectiveness of our approach.

2 Case study: airline crew assignment

Crew scheduling at airlines is usually divided into a crew pairing and a crew assignment (or rostering) phase. Firstly, anonymous pairings (or crew rotations) are formed out of the flight legs (flights without stopover) such that the crew needs on each flight are covered. Then in crew assignment, the pairings together with other activities such as ground duties, reserve duties and off-duty blocks are sequenced to rosters and assigned to individual crew members. In both problems, complex rules and regulations coming from legislation and contractual agreements have to be met by the solutions and some objective function has to be optimized. The airline crew assignment problem can be viewed as resource allocation problem where the activities to be assigned are fixed in time. In practice, around 100 complex rules and regulations have to be met by the created rosters, and additional constraints between rosters and/or crew members have to be taken into account. The problem is considered to be large scale, and in concrete applications, several thousand activities have to be assigned to up to 1000 crew members. In this paper, we apply our approach to real data of a major European airline.

The standard methods for solving crew assignment problems are based on the generate and optimize principle, i.e. on column generation. In the master problem, a set partitioning type problem is solved to select exactly one roster for each crew member such that the capacities of the activities are met, the solution satisfies constraints between several crew members, and the objective is optimized. Since it is not possible to have an explicit representation of all

possible rosters, the master problem is always defined on a subset of all possible rosters.

In the subproblem, a large number of legal rosters is generated. This is either done by partial enumeration based on propagation and pruning techniques [13], or by solving a constrained shortest path problem where the constraints ensure that only legal rosters are generated, and where the objective function is equivalent to 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 [8]. The latter approach is known as constrained shortest path column generation. In that approach the subproblem is solved optimally and one can prove that it is possible to obtain the optimal solution to the entire problem without explicit enumeration of all possible rosters. In either case one can iterate between the subproblem and the master problem.

Constrained shortest path column generation is a very powerful technique in terms of optimization potential, but efficient algorithms for the constrained shortest path problem do not permit arbitrary constraints. Therefore this approach is not compatible with all real-world rules and regulations. Using the framework presented below, we show how to overcome these limitations for the difficult problem of crew assignment. As a result, we maintain full expressiveness with respect to rules and regulations, and obtain the optimization benefits of the column generation approach.

3 A framework for CP based column generation

3.1 The general framework

In this section, we introduce a general framework for constraint programming based column generation where the master problem is a MIP. The novelty is that the subproblem can be an arbitrary CSP which has two major advantages. Firstly, it generalizes the class of subproblems and thus allows to use column generation even if the subproblem does not reduce to a MIP-problem. Secondly, it allows to exploit constraint satisfaction techniques to solve the subproblem. Constraint-based column generation is particularly well-suited for subproblems that can partially, but not entirely be solved by polynomial OR methods. In this case, some constraints do not fit and have to be treated separately. In the constraint satisfaction approach, the optimization algorithm, as well as the algorithms of the other constraints will be used in a uniform way, namely to reduce the domains of variables. We can also say that the CSP-approach allows different algorithms to communicate and to co-operate.

The basic idea of constraint programming based column generation is very simple. The master problem is a mixed integer problem which has a set of linear constraints and a linear cost function and the columns (or variables) of the master problem are not given explicitly. Without loss of generality, we assume that the objective is to be minimized. The subproblem is an arbitrary CSP. For each solution of the subproblem there exists a variable in the master problem. Of

course, we have to know the coefficients of the variable in all linear constraints of the master problem and in its linear cost function. For each of these coefficients $a_{i,j}$, we introduce a corresponding variable y_i in the subproblem. Furthermore, we introduce a variable z for the coefficient c_j in the cost function. Given a solution v_j of the subproblem, the coefficient $a_{i,j}$ of the variable x_j in the i -th linear constraint is then obtained as the value of the variable y_i in the given solution.

Representing coefficients by variables of the subproblem also allows to ensure that solutions of the subproblem have negative reduced costs. Given a solution of a linear relaxation of the master problem, we consider the dual values λ_i of each constraint i . We then simply introduce a linear constraint in the subproblem which is formulated on z and the y_i -s and which uses the dual values as coefficients.

We now introduce these ideas more formally. A constraint satisfaction problem is defined as follows:

Definition 1. Let $P := (\mathcal{X}, D, \mathcal{C})$ be a constraint satisfaction problem (CSP) where \mathcal{X} is a set of variables, D is a set of values, and \mathcal{C} is a set of constraints of the form $((x_1, \dots, x_n), R)$ where $x_i \in \mathcal{X}$ and $R \subseteq D^n$ is a relation. A mapping $v : \mathcal{X} \rightarrow D$ of the variables to the values of the domain satisfies a constraint $((x_1, \dots, x_n), R)$ of \mathcal{C} iff $(v(x_1), \dots, v(x_n))$ is an element of the relation R . A solution of P is a mapping $v : \mathcal{X} \rightarrow D$ that satisfies all constraints in \mathcal{C} .

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

A subproblem can be represented by an arbitrary CSP. A constraint of the master problem is represented by a variable of this subproblem, a sign, and a right-hand-side.

Definition 2. Let $SP := (\mathcal{X}, D, \mathcal{C})$ be a CSP. A master constraint for SP is a triple (y, s, b) where $y \in \mathcal{X}$, $s \in \{-1, 0, +1\}$ and b is arbitrary.

The master problem is then specified by a subproblem, a set of m master constraints, and a variable representing the cost coefficient of a column.

Definition 3. A master problem MP is specified by a triple (SP, \mathcal{M}, z) where SP is a CSP $(\mathcal{X}, D, \mathcal{C})$, $\mathcal{M} = \{mc_1, \dots, mc_m\}$ is a set of master constraints for SP and $z \in \mathcal{X}$ is a variable of SP .

Given a master problem and a set S of solutions of the subproblem, we define a mixed integer problem MIP representing MP as follows:

Definition 4. Let MP be a master problem as in Def. 3 and S be a set of solutions of the subproblem SP of MP . The MIP representing MP and S is defined as follows:

1. For each solution $v \in S$ of SP there exists a variable x_v .

2. For each master constraint $mc_i = (y_i, s_i, b_i)$ there exists a linear constraint of the following form

$$\begin{array}{lll} \sum_{v \in S} v(y_i) \cdot x_v \leq b_i & \sum_{v \in S} v(y_i) \cdot x_v = b_i & \sum_{v \in S} v(y_i) \cdot x_v \geq b_i \\ \text{for } s_i = -1 & \text{for } s_i = 0 & \text{for } s_i = 1 \end{array} \quad (1)$$

3. The objective is to minimize $\sum_{v \in S} v(z) \cdot x_v$.

(Again, without loss of generality we consider a minimization problem only.) An optimal solution of the linear relaxation of this MIP (plus optional branching decisions) produces dual values for the master constraint (y, s, b) . We can use them to add a negative reduced cost constraint to the subproblem:

Definition 5. Let λ_i be a dual value for the master constraint (y_i, s_i, b_i) . Then the negative reduced cost constraint (NRC) for these dual values has the variables (z, y_1, \dots, y_n) and is defined by the following condition:

$$\bar{z} - \sum_{i=1}^n \lambda_i \cdot \bar{y}_i \leq 0 \quad (2)$$

Although it is sufficient to generate arbitrary columns with negative reduced costs, columns with smaller reduced costs will lead to better solutions of the master problem. We therefore introduce a variable for the left-hand-side of (2) and we use it as objective for the subproblem.

Hence, we obtain a simple framework that defines constraint programming based column generations. A column of the master problem is represented by a solution of a subproblem. Furthermore, the coefficient of a column in a constraint is represented by a variable of the subproblem. Our framework is compatible with different methods for solving the master problem, e.g. the Branch-and-Price method where columns are generated in the search nodes of the master problem (cf. e.g. [1] for details).

3.2 Path optimization subproblems

In many applications of column generation, the subproblem reduces to a problem of finding a shortest path in a graph that satisfies additional constraints. We now show how to express this important special case in our framework.

For the sake of brevity, we limit our discussion to directed acyclic graphs. Let (V, E) be a directed acyclic graph where $V := \{0, \dots, n+1\}$ is the set of nodes. Let $e := |E|$ be the number of edges. We suppose that $s := 0$ is a unique source node, that $t := n+1$ is a unique sink node, and that the graph is topologically ordered¹ (i.e. $(i, j) \in E$ implies $i < j$).

We now suppose that the subproblem consists of finding a path through this graph that respects additional constraints. Furthermore, we suppose that there

¹ Each directed acyclic graph can be transformed into a graph of this form in time $O(n + e)$.

are master constraints that count how often a node occurs in a path. Given a solution of the subproblem, the coefficient of the corresponding column in such a constraint has the value 1 iff the considered node occurs in the selected path. For each node $i \in \mathcal{N} := \{1, \dots, n\}$, we therefore introduce a boolean variable y_i in the subproblem. This variable has the value 1 iff node i is contained in the selected path. Thus, the path is represented by an array of boolean variables. In some cases, it is also of advantage to represent it by a constrained set variable² Y . The value $v(Y)$ of this variable is a subset of \mathcal{N} . Given this set variable, we can define the boolean variables by the following constraint:

$$\overline{y}_i = 1 \text{ iff } i \in \overline{Y} \quad (3)$$

The cost coefficient of a variable in the master problem can often be expressed as costs of the selected path. We therefore suppose that edge costs $c_{i,j}$ are given for each edge $(i, j) \in E$. In order to determine the path costs, we need the next node of a given node $i \in \mathcal{N} \cup \{s\}$. For direct acyclic graphs, this next node is uniquely defined.³

$$next(i, Y) := \min(\{j \in \overline{Y} \mid j > i\} \cup \{t\}) \quad (4)$$

We now suppose that the cost variable z of the subproblem is the sum between the path costs and a remainder z' .

$$\overline{z} = \overline{z}' + \sum_{i \in \overline{Y} \cup \{s\}} c_{i, next(i, Y)} \quad (5)$$

We can also express the negative reduced cost constraint in this form. In addition to the boolean variable y_i , we can have n' variables y'_i for other kinds of master constraints. Let λ_i be the dual values for y_i and λ'_i be the dual values for y'_i . The dual values λ_i can immediately be subtracted from the original edge costs:

$$c'_{i,j} := \begin{cases} c_{i,j} & \text{if } i = s \\ c_{i,j} - \lambda_i & \text{otherwise} \end{cases} \quad (6)$$

The negative reduced cost constraint has then the form

$$\sum_{i \in \overline{Y} \cup \{s\}} c'_{i, next(i, Y)} + \left(\overline{z}' - \sum_{i=1}^{n'} \lambda'_i \cdot \overline{y}'_i \right) \leq 0 \quad (7)$$

The first part is the modified path costs. The second part treats the remaining costs and has the form of the usual negative reduced cost constraint. Below, we introduce a single constraint that ensures that Y represents a path and that also determines the cost of the path. It thus allows to encode the constraints above. This constraint needs only the set variable and a description of the graph. It avoids the introduction of additional variables for $next(i, Y)$ and $c_{i, next(i, Y)}$.

² Constrained set variables are supported by [11] and allow a compact representation of an array of boolean variables. Constraints on set variables such as sum-over-set constraints often allow a better and more efficient propagation than corresponding constraints on boolean variables.

³ For cyclic graphs, we have to introduce constraint variables for the next node.

4 An efficient path constraint on set variables

4.1 Semantics and propagation

In this section, we introduce an efficient path constraint for directed acyclic graphs with edge costs (so-called networks). The constraint ensures that a given set variable represents a path through this graph. Furthermore, the constraint ensures that a given variable represents the cost of this path. We also show how bound consistency can be achieved by determining shortest and longest paths.

The path constraint is defined for a directed acyclic graph (V, E) (of the same form as introduced in Sect. 3.2), the edge costs $c_{i,j}$, a set variable Y and a variable z . The constraint 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 t , i.e. $\bar{Y} \subseteq \mathcal{N}$ and

$$\{(i, next(i, Y)) \mid i \in \bar{Y} \cup \{s\}\} \subseteq E \quad (8)$$

2. z is the sum of the edge costs

$$\bar{z} = \sum_{i \in \bar{Y} \cup \{s\}} c_{i, next(i, Y)} \quad (9)$$

Compared to the existing path constraint of ILOG SOLVER 4.3 [11], the new path constraint is formulated on a set variable and can thus add arbitrary nodes of \mathcal{N} to a partially known path or remove them from it.

Next we discuss how to achieve bound consistency for the path constraint. We introduce lower and upper bounds for the variables z and Y . Let $min(z)$ be a lower bound for the value of z and $max(z)$ be an upper bound. Furthermore, let $req(Y)$ be a lower bound for the set variable Y . This set contains all the elements that must be contained in the path and is therefore called required set. Furthermore, let $pos(Y)$ be an upper bound for Y . This set is also called possible set since it contains the nodes that can possibly belong to the path.

We say that a path P is *admissible* (w.r.t. the given bounds) if it starts in the source s , terminates in the sink t , and $req(Y) \subseteq P \subseteq pos(Y)$. We say that a path P is *consistent* (w.r.t. the given bounds) if it is admissible and if the costs of the path are greater than $min(z)$ and smaller than $max(z)$. We say that the *bounds are consistent* if they satisfy the following conditions:

1. For each $i \in pos(Y)$ there exists a consistent path P through i and for each $i \notin req(Y)$ there exists a consistent path P that does not contain i .
2. There exist a consistent path with costs $max(z)$ and a consistent path with costs $min(z)$.

If the given bounds are not consistent then we can make them tighter using the following propagation rules.

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) then we replace it by lb . If the bound $\max(z)$ is greater than the costs ub of the longest admissible path (or $-\infty$ if there is no admissible path) then we replace it by ub .
2. If the costs of the shortest admissible path through i for $i \in \text{pos}(Y)$ are strictly greater than the upper bound $\max(z)$ then we can remove i from $\text{pos}(Y)$. If the costs of the longest admissible path through i for $i \in \text{pos}(Y)$ are strictly smaller than the lower bound $\min(z)$ then we can remove i from $\text{pos}(Y)$.
3. If the costs of the shortest admissible path that does not contain i for $i \notin \text{req}(Y)$ are strictly greater than the upper bound $\max(z)$ then we have to add i to $\text{req}(Y)$. If the costs of the longest admissible path that does not contain i for $i \notin \text{req}(Y)$ are strictly smaller than the lower bound $\min(z)$ then we have to add i to $\text{req}(Y)$.

Repeated application of these propagation rules will establish consistent bounds (or lead to an empty domain, i.e. an inconsistency). The propagation rules themselves require the detection of shortest (and longest) admissible paths. Some of them require that these paths contain a given node i , others require that the paths do not contain a node i . In the next section, we show how shortest paths satisfying these conditions can be computed efficiently. Furthermore, we discuss how to maintain these paths when incremental updates occur. Longest paths can be determined similarly by applying the algorithms to the negated edge costs $-c_{i,j}$ and to the negated cost variable $-y$.

4.2 Initial and incremental propagation algorithms

In order to achieve bound consistency for the path constraint, we have to find shortest paths from the source to all the nodes. Since we deal with directed acyclic graphs we can use a variant of Dijkstra's shortest path algorithm that visits nodes in topological order and thus runs in linear time $O(n + e)$. Furthermore, it does not require that edge costs are positive (cf. e.g. [5] for details).

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

For propagation rule 2 we must determine the cost of a shortest admissible path going through node $i \in \text{pos}(Y)$. This cost is simply the sum of the costs y_i^s of the shortest path from the sink to node i and the costs y_i^t of the shortest path from i to the sink. The latter can be computed with the same algorithm by just inverting all the edges and by applying it to the sink. If $y_i^s + y_i^t$ is strictly greater than $\max(z)$ we remove i from the possible set. Algorithm 1 shows the implementation of propagation rules 1 and 2.

For the sake of brevity, we omit the detailed algorithm for propagation rule 3 and outline only the idea. We remove all edges (i, j) if the costs of the shortest

Algorithm 1 Shortest Path with propagation rules 1 and 2

```
for all  $i \in V$  do
   $y_i^t := \infty; y_i^s := \infty; \pi_i := \text{NIL}; \sigma_i := \text{NIL};$  // Init
   $y_s^s := 0; y_t^t := 0; k^s := 0; k^t := 0$  // Init source and sink
  // determining shortest path from source  $s$  to all nodes
  for all  $i \in V$  taken in increasing topological order do
    if  $k^s \leq i$  then
       $k^s := \min\{l \in \text{req}(Y) \mid l > i\};$ 
      for all  $j \in \text{pos}(Y)$  s.t.  $(i, j) \in E$  and  $j \leq k^s$  do
        if  $y_j^s > y_i^s + c_{i,j}$  then
           $y_j^s := y_i^s + c_{i,j}; \pi_j := i;$ 
      // determining reverse shortest path from sink  $t$  to all nodes
    for all  $i \in V$  taken in decreasing topological order do
      if  $k^t \geq i$  then
         $k^t := \max\{l \in \text{req}(Y) \mid l < i\};$ 
        for all  $j \in \text{pos}(Y)$  s.t.  $(j, i) \in E$  and  $j \geq k^t$  do
          if  $y_j^t > y_i^t + c_{j,i}$  then
             $y_j^t := y_i^t + c_{j,i}; \sigma_j := i;$ 
      if  $y_i^s > \min(z)$  then
         $\min(z) := y_i^s$  // propagation rule 1
      for all  $i \in \text{pos}(Y)$  do
        if  $y_i^s + y_i^t > \max(z)$  then
           $\text{pos}(Y) := \text{pos}(Y) \setminus \{i\}$  // propagation rule 2
```

path through this edge are strictly greater than $\max(z)$. We then determine the cut nodes of the resulting graph and add them to $\text{req}(Y)$. The edge removal and the cut point detection can be achieved in time $O(n + e)$ which allows us to state the following theorem.

Theorem 1. *Bound consistency for the path constraint on a directed acyclic graph can be computed in time $O(n + e)$.*

We are also interested in how to maintain bound consistency efficiently and we suppose that we can use the AC5-algorithm [10] as implemented in ILOG SOLVER as a framework for this. We thus obtain changes of the domains of z and Y (increase of $\min(z)$; decrease of $\max(z)$; elements added to $\text{req}(Y)$; elements removed from $\text{pos}(Y)$). We detail only the last event. Interestingly, the shortest path algorithm already provides a notion of current support that allows to implement an incremental algorithm in the style of AC6 [3]. Each node i has a current predecessor π_i and a current successor σ_i . We have to update the costs y_i^s if the the current predecessor π_i is removed from the possible set or the costs $y_{\pi_i}^s$ have been changed. In order to achieve this, we introduce a list of nodes that are currently supported by a given node. If k is removed from the possible set we visit all nodes i supported by k and update their costs. If the costs change then we apply propagation rule 2 to i . Furthermore, we repeat the procedure for the nodes supported by i . Further work is needed to elaborate the details of this procedure and to check whether propagation rule 3, as well as the other

events can be treated similarly. Moreover, we will analyze the precise complexity of maintaining bound consistency.

5 Application to crew assignment

5.1 The subproblem of column generation

The constraints of the crew assignment problem are formulated in the PARROT roster library which provides a modeling layer on top of ILOG SOLVER. This layer facilitates the expression of complex crew regulations and translates them into constraints on set variables describing the activities of a crew member and on integer variables describing derived attributes of crew members and activities.

The roster library introduces new constraints on set variables such as a sum-over-set-constraint, a next- and a previous-constraint, and a gliding-sum constraint. The previous constraint can, for example, be used to define the rest time before an activity as the difference between its start time and the end time of the previous activity. The gliding-sum constraint ensures that the amount of flight time of a crew member in time windows of a given length does not exceed a maximal flight time. Those constraints ensure bound consistency for the set and integer variables.

When we generate rosters for a selected crew member we additionally set up a legality graph between the possible activities of this crew member and post the path constraint on his/her set variable. The graph describes the possible successions between activities as established by roster library propagation. An activity j can be a direct successor of an activity i if we do not get an inconsistency when assigning i and j to the crew member and when removing all activities between i and j from her/him.

The generation process is usually iterative, i.e. after generating a certain number of rosters, the master problem is solved. The duals are then used to generate new rosters, and so on. We point out, that in the application to crew assignment, we usually do not encounter a gap between the final continuous linear programming relaxation of the master problem, thus we can prove optimality in the present case. We also note that in most cases, the cost structure of crew assignment problems meets the assumption of Sect. 3.2.

5.2 Numerical results

In this section, we present computational results on real test data of a major European airline. The considered test cases consist of typical selections of the rules and regulations from the production system of the airline. These, together with crew information and the activities to be assigned, form the so called Large-Size-Airline-Case in PARROT. In the present paper, we use a simplified but representative objective function and we do not consider constraints concerning more than one roster which means that the master problem is a set partitioning problem. However, from our experience with the data we consider the obtained results as representative.

In the following we highlight some of the findings of the experiments. Thereby we characterize an instance by the number of crew members, the number of preassigned activities, and the number of activities to be assigned. For example, an instance of type 10-16-40 consists of 10 crew members, 16 preassignments, and 40 activities.

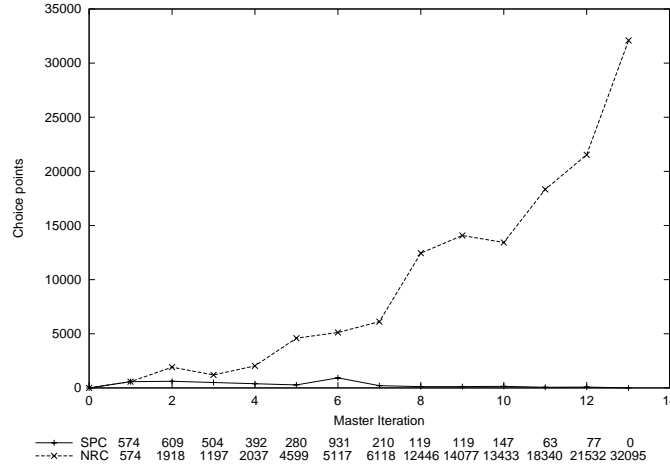


Fig. 1. Number of choice points for NRC and SPC for an instance of type 7-0-30.

Firstly, we compare the number of choice points considered when using the shortest path constraint (SPC) with the propagation as described in Sect. 4 and with the use of the “pure” NRC as defined in Def. 5. The result for a small instance of type 7-0-30, where one can still prove optimality using NRC only, is shown in Fig. 1. The pruning effect of SPC allows to prove optimality in iteration 13 without considering any choice points, while it becomes very expensive to prove optimality when using NRC only. Figure 2 presents the pruning behavior of SPC for instances of type 40-100-199 and 65-165-250 which is of the same type as for the smaller instance.

The reduction of choice points does not automatically yield a better computation time, as the time spent per choice point tends to be higher. However, Fig. 3 shows that the latter is not the case. The left plot shows a comparison of two program versions, one using SPC and one using NCR only, in a time versus quality diagram. Although the SPC version needs more time in the beginning (due to the more costly initial propagation), it catches up quickly and performs much better than the version not using it. In Fig. 3 we also compare the behavior of IP-costs and LP-costs in the right diagram. As mentioned above, the gap between IP and LP diminishes, i.e. it is possible to prove optimality. We also performed experiments on how well SPC scales. The results showed that the pruning effect of SPC gets better for growing input size.

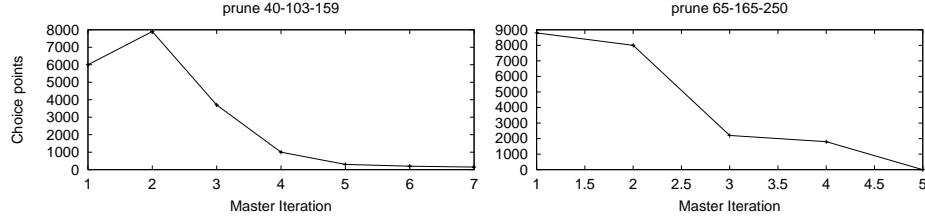


Fig. 2. Pruning for SPC for instances of type 40-100-199 and 65-165-250.

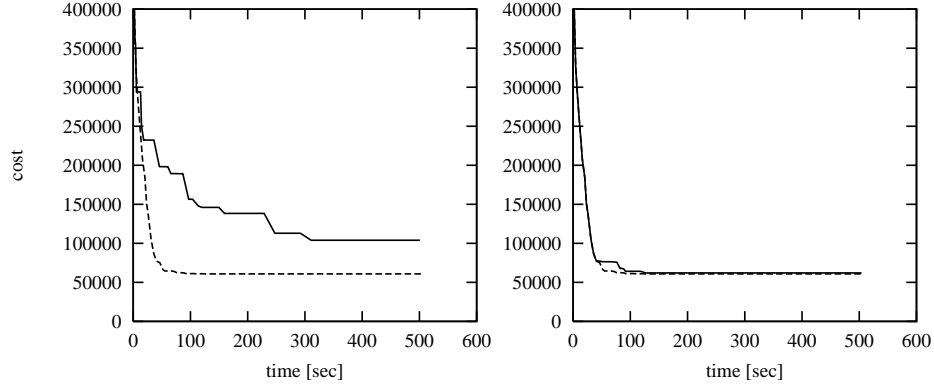


Fig. 3. The left picture shows time versus quality with and without SPC and the right picture the development of LP- and IP-costs with SPC on an instance of type 10-16-40.

To give two concrete examples for running times for solving problems to optimality using SPC: an instance of the type 40-103-199 took 1921 seconds and an instance of the type 65-165-250 took 3162 seconds of CPU time on a Sun Ultra Enterprise 450 with 300 MHz. These running times are encouraging, considering the early stage of the development of the different software components used to perform the computations.⁴

6 Conclusions

In this paper, we introduced and applied a framework for constraint programming based column generation, which is a new way for integrating constraint programming and linear programming. This allows to tackle large-scale optimization problems of a certain structure. Compared to traditional methods of column generation, we formulate the subproblem as a CSP and thus extend the modeling facilities of column generation. Compared to co-operative solvers (cf.

⁴ In PARROT, these software components are currently further developed and significant performance improvements can be expected.

e.g. [2, 4, 15]), the CP and LP solver do not communicate only by reducing domains, but mainly by exchanging solutions and dual values. The use of the duals in the negative reduced cost constraint reduces the domains for the next solution. Optimization methods that are usually used for solving the subproblem can be encapsulated in a global constraint. We demonstrated this for shortest path problems and developed a path constraint on set variables for networks that achieves bound consistency in linear time. This new path constraint has been a key element for optimally solving difficult airline crew assignment problems. We have thus demonstrated that constraint programming can indeed increase the power of the column generation method.

References

1. C. Barnhart, E.L. Johnson, G.L. Nemhauser, M.W.P. Savelsbergh, and P.H. Vance. Branch-and-Price: Column Generation for Huge Integer Programs. In *Operations Research*, 46:316-329, 1998.
2. 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*, pages 245-272. Elsevier, 1995.
3. C. Bessière. Arc-consistency and arc-consistency again. *Artificial Intelligence*, 65:179-190, 1994.
4. A. Bockmayr and T. Kasper. Branch-and-Infer: A unifying framework for integer and finite domain constraint programming. *INFORMS Journal of Computing*, 10(3):287-300, 1998.
5. T.H. Cormen, C.E. Leieron, R.L. Riverste. *Introduction to Algorithms*, McGraw-Hill, 1990.
6. G.B. Dantzig and P. Wolfe. The decomposition algorithm for linear programs. *Econometrica*, 29(4):767-778, 1961.
7. J. Desrosiers, M.M. Solomon, and F. Soumis. Time constrained routing and scheduling. In *Handbooks of Operations Research and Management Science*, 8:35-139, 1993.
8. F. Gamache, F. Soumis, D. Villeneuve, J. Desrosiers, and E. Gélina. The preferential bidding system at Air Canada. *Transportation Science*, 32(3):246-255, 1998.
9. P.C. Gilmore and R.E. Gomory. A linear programming approach to the cutting stock problem. *Operations Research*, 9:849-859, 1961.
10. P. Van Hentenryck, Y. Deville, and C.M. Teng. A generic arc-consistency algorithm and its specializations. *Artificial Intelligence*, 57:291-321, 1992.
11. ILOG. ILOG SOLVER. Reference manual and user manual. V4.3, ILOG, 1998.
12. E. Jacquet-Lagrange and M. Lebar. Column generation for a scheduling problem with maintenance constraints. In *CP'98 Workshop on Large-Scale Combinatorial Optimization and Constraints*, Pisa, Italy, 1998.
13. N. Kohl and S.E. Karisch. Airline crew assignment: modeling and optimization. Carmen Report, 1999. In preparation.
14. PARROT. Executive Summary. ESPRIT 24960, 1997.
15. 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.