Hybrid Column Generation Approaches for Solving Real World Crew Management Problems

Tallys H. Yunes *Arnaldo V. Moura*
Cid C. de Souza

Relatório Técnico IC–00-18

Dezembro de 2000

Hybrid Column Generation Approaches for Solving Real World Crew Management Problems[*]

Tallys H. Yunes Arnaldo V. Moura Cid C. de Souza

Abstract

This article considers the overall crew management problem that arises from the daily operation of an urban transit bus company that serves the metropolitan area of the city of Belo Horizonte, in Brazil. Due to its intrinsic complexity, the problem is divided in two distinct problems, namely: *crew scheduling* and *crew rostering*. We have tackled each one of these problems using Mathematical Programming (MP) and Constraint Logic Programming (CLP) approaches. Besides, we also developed hybrid column generation algorithms for solving these problems, combining MP and CLP. The hybrid algorithms always performed better, when obtaining optimal solutions, than the two previous isolated approaches. In particular, it proved much faster for the scheduling problem. All the proposed algorithms have been implemented and tested over real world data obtained from the aforementioned company. The coefficient matrix of the linear program associated with some instances of the scheduling problem contains tens of millions of columns, and this number is even larger for the rostering problem. The analysis of our experiments indicates that it was possible to find high quality, and many times optimal, solutions that were suitable for the company's needs. These solutions were obtained within reasonable computational times, on a typical desktop PC.

Introduction

The overall *crew management* problem concerns the allocation of trips to crews within a certain planning horizon. In addition, it is necessary to respect a specific set of operational constraints and minimize a certain objective function. Being a very hard problem, when taken in its entirety, it is usually divided in two smaller problems: the *crew scheduling* problem and the *crew rostering* problem (see [6]). In the crew scheduling problem, the aim is to partition the initial set of trips into a minimal set of *feasible duties*. Each such duty is an ordered sequence of trips which is to be performed by the same crew and that satisfies a subset of the original problem constraints: those related to the sequencing of trips during a workday. The crew rostering problem takes as input the set of duties output by the crew scheduling phase and builds a roster spanning a longer period, e.g. months or years. In the latter case, the roster must satisfy a different set of constraints: those related to rest periods, vacations and other long term operational restrictions.

[*]Research supported by FAPESP (grants 97/10982-0 and 98/05999-4), CNPq (300883/94-3), FINEP (ProNEx 107/97), and CAPES. Submitted for publication.

This article describes the crew management problem stemming from the operation of a Brazilian bus company that serves a major urban area in the city of Belo Horizonte. This area serves more than two million inhabitants, in central Brazil. Since employee wages may well rise to 50 percent or more of the company's total expenditures, even small percentage savings can be quite significant. The related crew scheduling and crew rostering problems are solved by means of hybrid column generation approaches involving both Integer Programming (IP) and Constraint Logic Programming (CLP) techniques. We also present pure IP and CLP solutions for these problems.

We started with the crew scheduling problem, applying a pure IP formulation, and using a classical branch-and-bound technique to solve the resulting set partitioning problem. Since this method requires that all feasible duties are previously inserted into the problem formulation, all memory resources were rapidly consumed when we reached half a million feasible duties. To circumvent this difficulty, we implemented a column generation technique. As suggested by [13], the subproblem of generating feasible duties with negative reduced cost was transformed into a constrained shortest path problem over a directed acyclic graph and then solved using Dynamic Programming techniques. However, due to the size and idiosyncrasies of our real problem instances, this technique did not make much progress towards solving large instances.

In parallel, we also implemented a heuristic algorithm presented in [4] which produced very good results on some related large set covering problems. With this implementation, problems with up to two million feasible duties could be solved to optimality. But this particular heuristic also requires that all feasible duties be present in memory during execution. Although some progress with respect to time efficiency was achieved, memory usage was still a formidable obstacle.

The difficulties we faced when using the previous approaches almost disappeared when we turned to a language that supports constraint specification over finite domain variables. We were able to develop and implement our models in a short time, producing code that was both concise and clear. When executed, it came as no surprise that the model showed two distinct behaviors, mainly due to the huge size of the search space involved. It was very fast when asked to compute new feasible duties, but lagged behind the IP methods when asked to obtain a provably optimal schedule. The search spaces of our problem instances are enormous and there are no strong local constraints available to help the resolution process. Also a good heuristic to improve the search strategy does not come easily, as noted in [12].

To harness the capabilities of both the IP and CLP techniques, we resorted to a hybrid approach to solve the larger, more realistic, problem instances. The main idea is to use the linear relaxation of a smaller core problem in order to efficiently compute good lower bounds on the optimal solution value. Using the values of dual variables present in the solution of the linear relaxation, we can enter a generation phase that computes new feasible duties. This phase is modeled as a constraint satisfaction problem that searches for new feasible duties with negative reduced costs. This problem is submitted to the constraint solver, which returns new feasible duties. After introducing these new duties into the IP problem formulation, the initial phase can be taken again, restarting the cycle. When the CLP solver announces the inexistence of new feasible duties with negative reduced cost, the optimality of the current solution is proved. This algorithm secures the strengths of both the pure IP and the pure CLP approaches: only a small subset of all the feasible duties is efficiently dealt with at a time, and new feasible duties are quickly computed only when they will make a difference. The resulting code was tested on some large instances, based on real data. As of this writing, we can solve, in a reasonable time and with proven optimality, instances of the crew scheduling problem with an excess of 150 trips and 12 million feasible duties.

Some quite specific union regulations and operational constraints make our rostering problem fairly distinct from some other known crew rostering problems found in the literature as [7] and [5]. In general, it is sufficient to construct one initial roster consisting of a feasible sequencing of the duties that spans the least possible number of days. The complete roster is then built by just assigning shifted versions of that sequence of duties to each crew so as to have every duty performed in each day of the planning horizon. In other common cases such as [19], [8] and [3], the main concern is to balance the workload among the crews involved. Although we also look for a roster with relatively balanced workloads, these approaches will not, in general, find the best solution for our purposes. We are not interested in minimizing the number of days needed to execute the roster, since the length of the planning horizon is fixed in advance. Our objective is to use the minimum number of crews when constructing the roster for the given period. Another difficulty comes from the fact that some constraints behave differently for each crew, depending on the amount of work assigned to it in the previous month. Moreover, different crews have different needs for days off, imposed by personal requirements.

Similarly to the crew scheduling problem, we started with models based on pure IP and CLP techniques to solve the rostering problem. Again, we also developed a hybrid column generation approach for this problem, which follows the same basic ideas of the one applied in the crew scheduling phase.

This article is organized as follows. Section 1 describes the crew scheduling problem, including a number of subsections. In Sect. 1.4, we discuss an IP approach and report on the implementation of two alternative techniques: standard column generation and heuristics. In Sect. 1.5, we investigate a pure CLP approach and, in Sect. 1.6, we present the hybrid approach. All the previous three sections report implementation details and computational results on real data sets. Section 2 gives a detailed description of the crew rostering problem. Its subsections present the different solution techniques that were investigated. Section 2.4 explains the format of the input data sets used in our experiments. In Sect. 2.5, we present an IP formulation of the problem, together with some computational results. A pure CLP model for the rostering problem is described in Sect. 2.6, where some experiments are also conducted to evaluate its performance. The results achieved with a hybrid column generation approach appear in Sect. 2.7. Finally, we draw the main conclusions and discuss further issues in Sect. 3.

All computation times presented in this text are given in CPU seconds of a Pentium II 350 MHz with 320 MB of RAM. Execution times inferior to one minute are reported as *ss.cc*, where *ss* denotes seconds and *cc* denotes hundredths of seconds. For execution times that exceed 60 seconds, we use the alternative notation *hh:mm:ss*, where *hh*, *mm* and *ss* represent hours, minutes and seconds, respectively.

1 The Crew Scheduling Problem

In a typical crew scheduling problem, a set of trips has to be assigned to some available crews. The goal is to assign a subset of the trips to each crew in such a way that no trip is left unassigned. As usual, not every possible assignment is allowed since a number of constraints must be observed. Additionally, a cost function has to be minimized.

1.1 Terminology

Among the following terms, some are of general use, while others reflect specifics of the transportation service for the urban area where the input data came from. A *relief point* is a location where crews may change buses and rest. The act of driving a bus from one relief point to another relief point, passing by no intermediate relief point, is named a *trip*. Associated with a trip we have its *start time*, its *duration*, its *departure relief point*, and its *arrival relief point*. The duration of a trip is statistically calculated from field collected data, and depends on many factors, such as the day of the week and the start time of the trip along the day. A *duty* is a sequence of trips that are assigned to the same crew. By *idle time* we denote any of the time intervals between two consecutive trips in a duty. Whenever this idle time exceeds *Idle_Limit* minutes, it is called a *long rest*. A duty that contains a long rest is called a *split-shift duty* or simply a *split shift*. The *rest time* of a duty is the sum of its idle times, not counting long rests. The parameter *Min_Rest* gives the minimum amount of rest time, in minutes, that each crew is entitled to. The sum of the durations of the trips in a duty is called its *working time*. The sum of the *working time* and the *rest time* gives the *total working time* of a duty. The parameter *Workday* is specified by union regulations and limits the daily total working time.

1.2 Input Data

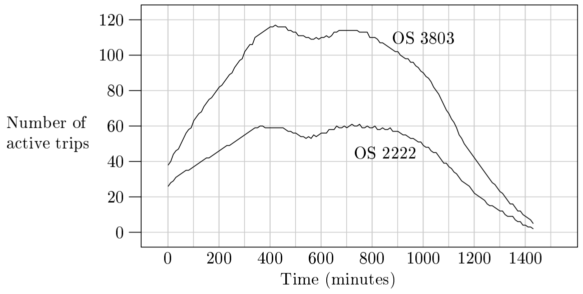
The input data comes in the form of a two dimensional table where each row represents one trip. For each trip, the table lists: *start time*, measured in minutes after midnight, *duration*, measured in minutes, *initial relief point* and *final relief point*. We have used data that reflect the operational environment of two bus lines, Line 2222 and Line 3803, that serve the metropolitan area around the city of Belo Horizonte, in central Brazil. Line 2222 has 125 trips and one relief point and Line 3803 has 246 trips and two relief points. The input data tables for these lines are called OS 2222 and OS 3803, respectively. Table 1(a) shows the first 10 rows of OS 3803. By considering initial segments taken from these two tables, we derived several other smaller problem instances. For example, taking the first 30 trips of OS 2222 gave us a new 30-trip problem instance. A measure of the number of active trips along a typical day, for both Line 2222 and Line 3803, is shown in Table 1(b). This figure was constructed as follows. For each (x, y) entry, we consider a time window $T = [x, x + \mathit{Workday}]$. The ordinate y indicates how many trips there are with start time s and duration d such that $s \in T$ or $s + d \in T$, i.e., how many trips are active in T.

Table 1: (a) Sample from OS 3803 (b) Distribution of trips along the day

(a)

Start	Dur	I. dep.	F. dep.
1	38	1	2
50	40	2	1
90	38	1	2
130	38	2	1
170	38	1	2
210	38	2	1
250	39	1	2
290	38	2	1
285	45	1	2
335	45	2	1

(b)



1.3 Constraints

For a duty to be feasible, it has to satisfy constraints imposed by labor contracts and union regulations, among others. For each duty we must observe

$$\begin{aligned} \textit{total working time} &\leq \textit{Workday} \\ \textit{rest time} &\geq \textit{Min\_Rest}\,. \end{aligned}$$

In each duty and for each pair (i, j) of consecutive trips, where i precedes j, we must have

$$\begin{aligned} (\textit{start time})_i + (\textit{duration})_i &\leq (\textit{start time})_j \\ (\textit{final relief point})_i &= (\textit{initial relief point})_j\,. \end{aligned}$$

Also, at most one long rest is allowed in each duty.

Restrictions from the operational environment impose *Idle_Limit* $= 120$, *Workday* $=$ 440, and *Min_Rest* $= 30$, measured in minutes. A *feasible duty* is a duty that satisfies all problem constraints. A *schedule* is a set of feasible duties and an *acceptable schedule* is any schedule that partitions the set of all trips. Since the problem specification treats all duties as indistinguishable, every duty is assigned a unit cost. The cost of a schedule is the sum of the costs of all its duties. Hence, minimizing the cost of a schedule is the same as minimizing the number of crews involved in the solution or, equivalently, the number of duties it contains. A *minimal schedule* is any acceptable schedule whose cost is minimal.

1.4 Mathematical Programming Approaches

Let m be the number of trips and n be the total number of feasible duties. The pure IP formulation of the problem is:

$$\min \sum_{j=1}^{n} x_j \tag{1}$$

$$\text{subject to} \quad \sum_{j=1}^{n} a_{ij} x_j = 1, \quad i = 1, 2, \ldots, m \tag{2}$$

$$x_j \in \{0, 1\}, \quad j = 1, 2, \ldots, n\,. \tag{3}$$

The x_j's are 0–1 decision variables that indicate which duties belong to the solution. The coefficient a_{ij} equals 1 if duty j contains trip i, otherwise, a_{ij} is 0. This is a classical set partitioning problem where the rows represent all trips and the columns represent all feasible duties.

We developed a constraint program to count all feasible duties both in OS 2222 and in OS 3803. Table 2 summarizes the results for increasing initial sections (column "# Trips") of the input data. The time (column "Time") needed to count the number of feasible duties (column "# FD") is also presented. For OS 2222, we get in excess of one million feasible duties, and for OS 3803 we get more than 122 million feasible duties.

It would be possible to adopt a set covering formulation if we replaced the '=' sign by a '$\geq$' sign in (2). In practice, this results in having crews riding on buses just like ordinary passengers. Despite the fact that a less expensive solution could arise from the set covering model, the latter was not used in practice since it may bring difficulties to the operational control.

1.4.1 A Pure Integer Programming Approach

In the pure IP approach, we used a constraint program to generate an output file containing all feasible duties. A program was developed in C to make this file conform to the CPLEX input format (CPLEX is a registered trademark of ILOG, Inc.). The resulting file was fed into a CPLEX LP solver. The node selection strategy used was *best-first* and branching was done upon the most fractional variable. Every other setting of the branch-and-bound algorithm used the standard default CPLEX configuration.

The main problem with the IP approach is clear: the number of feasible duties is enormous. Computational results for OS 2222 appear in Table 3, columns under "Pure IP". In that table, columns "Opt" and "Sol" indicate, respectively, the optimal and computed values for the corresponding run. It soon became apparent that the pure IP approach using the CPLEX solver would not be capable of obtaining the optimal solution for the complete OS 2222 problem instance. Besides, memory usage was also increasing at an alarming pace, and execution time was lagging behind when compared to other approaches that were being developed in parallel. As an alternative, we decided to implement a column generation approach.

Table 2: Number of feasible duties for OS 2222 and OS 3803

OS 2222 (1 relief point)			OS 3803 (2 relief points)		
# Trips	# FD	Time	# Trips	# FD	Time
10	63	0.07	20	978	1.40
20	306	0.33	40	6,705	5.98
30	1,032	0.99	60	45,236	33.19
40	5,191	5.38	80	256,910	00:03:19
50	18,721	21.84	100	1,180,856	00:18:34
60	42,965	00:01:09	120	3,225,072	00:57:53
70	104,771	00:03:10	140	8,082,482	02:59:17
80	212,442	00:05:40	160	18,632,680	08:12:28
90	335,265	00:07:48	180	33,966,710	14:39:21
100	496,970	00:10:49	200	54,365,975	17:55:26
110	706,519	00:14:54	220	83,753,429	42:14:35
125	1,067,406	01:00:27	246	122,775,538	95:49:54

1.4.2 Column Generation with Dynamic Programming

Column generation is a technique that is widely used to handle linear programs which have a very large number of columns in the coefficient matrix (see [1]). The method works by repeatedly executing two phases. In a first phase, instead of solving a linear relaxation of the whole problem, in which all columns are required to be loaded in memory, we quickly solve a smaller problem, called the *master* problem, that deals only with a subset of the original columns. That smaller problem solved, we start phase two, looking for columns with negative reduced costs. If there are no such columns, we have proved that the solution at hand indeed minimizes the objective function. Otherwise, we augment the master problem by bringing in a number of columns with negative reduced cost, and start over on phase one. From the pure IP formulation above, the reduced cost of a feasible duty d is given by $1 - \sum_{j \in T} u_j$, where T is the set of trips contained in d and u_j is the value of the dual variable associated with trip j. The problem of computing columns with negative reduced costs is called the *slave* subproblem. When the original variables have integer values, this algorithm must be embedded in a branch-and-bound strategy. The resulting algorithm is also known as *branch-and-price*.

Initializing. In order to start the algorithm, one has to decide how to setup the first master problem. According to a general guideline from [29], one should avoid trivial columns and also some apparently good initial collection of columns that may cause the method to wander into unpromising regions. In our case, however, a trivial initialization worked best. In an attempt to achieve a better performance, we augmented the initial identity matrix with a set of columns computed using the constraint program discussed in Sect. 1.5.1.

Table 3: Computational results for OS 2222 (1 relief point)

			Pure IP		CG+DP		Heuristic	
# Trips	# FD	Opt	Sol	Time	Sol	Time	Sol	Time
10	63	7	7	0.02	7	0.01	7	0.05
20	306	11	11	0.03	11	0.07	11	0.30
30	1,032	14	14	0.06	14	0.52	14	10.37
40	5,191	14	14	3.04	14	9.10	14	13.02
50	18,721	14	14	14.29	14	00:01:29	14	00:30:00
60	42,965	14	14	00:01:37	14	00:07:54	14	00:30:22
70	104,771	14	14	00:04:12	14	00:44:19	14	00:03:28
80	212,442	16	16	00:33:52	16	03:53:58	16	00:16:24
90	335,265	18	18	00:50:28	18	08:18:53	18	00:22:42
100	496,970	20	20	02:06:32	20	15:08:55	20	00:50:01
110	706,519	22	-	-	-	-	22	01:06:17
125	1,067,406	25	-	-	-	-	25	01:55:12

Computational results did not favor this alternative and we refrained from using it in subsequent experiments. The master problems were solved using the CPLEX LP solver.

Generating Columns. In general, the slave subproblem can also be formulated as another IP problem. In our case, constraints like the one on split-shift duties substantially complicate the formulation of a pure IP model. As another approach, [13] suggest reducing the slave subproblem to a constrained shortest path problem, formulated over a related directed acyclic graph G. When the algorithm for solving the slave problem is about to start, the value of all the dual variables can be easily extracted from the linear relaxation solution of the current master problem. For each trip i, we include in G two nodes, S_i and E_i, representing the start and end times of i, respectively, and an arc called a *trip arc* from S_i to E_i. Each trip arc is assigned a cost u_i, which is the same as the current value of the dual variable associated with trip i. An arc with cost zero connects the end vertex of a trip i to the start vertex of a trip j whenever the end time of i precedes the start time of j. Also, zero cost arcs connect a source node s to the start vertices of all trips, and some other zero cost arcs connect the end vertices of all trips to a sink node t. In this way, a path p from s to t in G represents a duty D, and the cost associated to p is the sum $\sum_{i \in D} u_i$, since only trip arcs in p have nonzero costs. From the IP formulation, we know that the reduced cost of a duty D is given by $1 - \sum_{i \in D} u_i$. Hence, to obtain a duty with negative reduced cost we seek a path in G whose associated cost is greater than 1. But we also need to guarantee that such a path represents a feasible duty. To this end, the algorithm keeps track of the resource consumption of each path it is dynamically constructing. When the next trip arc is added to a path, the latter becomes infeasible if this trip arc depletes any resource beyond its limits. If the path remains feasible, the resources consumed by the new trip arc adjoined to the path are subtracted from their respective current values, its cost is added to the present cost of the path, and the algorithm resumes looking for the next trip arc. This cycle terminates when the sink node is reached. In our case, besides the cost, we used three resources representing the total working time, the total rest time and a binary value that indicates if the path stands for a split-shift duty. To guarantee that the whole path can be reconstructed when the final node t is reached, a backward pointer is also maintained at each node. Using $-u_i$ as the cost associated to trip arc i, a dynamic programming algorithm can be implemented to compute a constrained shortest path in G. Since different paths consume resources in different amounts, the implementation is further complicated because it is necessary to maintain, at each node, a list of feasible paths that can reach that node from s. A path that reaches a node can only be discarded if it is disadvantageous, in terms of the consumption of *all* resources, with respect to another path that also reaches that same node. When this process terminates, however, it is easy to extract not only the shortest feasible path, but also a number of additional feasible paths, all with negative reduced costs. We complemented these ideas with additional improvements from [2] and our own experience.

Implementation and Results. To implement the branch-and-price strategy, the use of the ABACUS branch-and-price framework saved a lot of programming time (ABACUS is a registered trademark of OREAS GmbH). One of the important issues was the choice of the branching rule. When applying a branch-and-bound algorithm to set partitioning problems, a simple branching rule is to choose a binary variable and set it to 1 on one branch and set it to 0 on the other branch, although [29] shows that there are situations where this might not be the best choice. This simple branching rule produced a very small number of nodes in the implicit enumeration tree (41 in the worst case). Hence, we judged that any possible marginal gains did not justify the extra programming effort required to implement a more elaborated branching rule such as the one developed by [26]. In Table 3, columns under "CG+DP", show the computational results for OS 2222. This approach did not reach a satisfactory time performance, mainly because the constrained shortest path subproblem is relatively loose. As a pseudo-polynomial algorithm, the state space at each node has the potential of growing exponentially with the input size. The number of feasible paths that the algorithm has to maintain became so large that the time spent looking for columns with negative reduced cost is responsible for more than 90% of the total execution time, on the average, over all instances. Table 4 supports this observation.

1.4.3 A Heuristic Approach

Heuristics offer another approach to solve crew scheduling problems and there are many possible variations. Initially, we set aside those heuristics that were unable to reach an optimal solution. As a promising alternative, we decided to implement the set covering heuristic developed by [4]. This heuristic won the FASTER competition jointly organized by the Italian Railway Company and AIRO, solving, in a reasonable time, large set covering problems arising from crew scheduling. Using our own experience and additional ideas from the chapter on Lagrangian Relaxation in [25], an implementation was written in C

Table 4: Pricing time for the branch-and-price algorithm in Section 1.4.2 over OS 2222

# Trips	Pricing Time	Total Time	$\frac{\text{Pricing Time}}{\text{Total Time}}$ %
20	0.04	0.07	57.1
30	0.43	0.52	82.7
40	8.82	9.10	96.9
50	00:01:26	00:01:29	96.9
60	00:07:45	00:07:54	98.2
70	00:43:58	00:44:19	99.2
80	03:53:06	03:53:58	99.6
90	08:18:11	08:18:53	99.9
100	15:07:22	15:08:55	99.8

Table 5: Heuristic over OS 3803 (2 relief points)

# Trips	# FD	Opt	Sol	Time
20	978	6	6	0.35
40	6,705	13	13	3.60
60	45,236	15	15	52.01
80	256,910	15	15	00:08:11
100	1,180,856	15	15	00:13:51
110	2,015,334	15	15	00:23:24

and went through a long period of testing and benchmarking. Tests executed on set covering instances coming from the OR-Library showed that our implementation is competitive with the original implementation in terms of solution quality. When this algorithm terminates, it also produces a lower bound for the optimal covering solution, which could be used as a bound for the partitioning problem as well. We verified, however, that on the larger instances, the solution produced by the heuristic turned out to be a partition already. Computational results for OS 2222 appear in Table 3, columns under "Heuristic". Comparing all three implementations, it is clear that the heuristic approach produced the best results. However, applying this heuristic to the larger OS 3803 data set was problematic. Since storage space has to be allocated to accommodate all feasible columns, memory usage becomes prohibitive. It was possible to solve instances with up to 2 million feasible duties, as indicated in Table 5. Beyond that limit, 320 MB of main memory were not enough for the program to terminate.

1.5 A Constraint Logic Programming Approach

Modeling with finite domain constraints is rapidly gaining acceptance as a promising programming environment to solve large combinatorial problems. This led us to model the crew scheduling problem using pure Constraint Logic Programming (CLP) techniques as well. All models described in this section were formulated using the ECL^{i}PS^e syntax (`http://www.icparc.ic.ac.uk/eclipse`). Due to its large size, the ECL^{i}PS^e formulation for each run was obtained using a program generator that we developed in C.

A simple pure CLP formulation was developed first. It used a list of items, each item being itself a list describing an actual duty. A number of recursive predicates guaranteed that each item would satisfy all labor and regulation constraints (see Sect. 1.3), and also enforced restrictions of time and relief point compatibility between consecutive trips. These feasibility predicates iterated over all list items. The database contained one fact for each line of input data, as explained in Sect. 1.2. The resulting model was very simple to program in a declarative environment. The formulation, however, did not reach satisfactory results when submitted to the ECL^{i}PS^e solver, as shown in Table 6, columns under "First Model". A number of different labeling techniques, different clause orderings and several variants on constraint representation were explored, to no avail. When proving optimality, the situation was even worse. It was not possible to prove optimality for instances with only 10 trips in less than an hour of execution time. The main reason for this poor performance may reside on the recursiveness of the list representation, and on the absence of reasonable lower and upper bounds on the value of the optimal solution which could aid the solver to discard unpromising labelings.

1.5.1 An Improved Model

The new model is based on a two dimensional matrix X of integers. The number of columns (rows) in X, *UBdutyLen* (*UBnumDut*), is an upper bound on the size of any feasible duty (the total number of duties). To calculate *UBdutyLen*, we start by summing up the durations of the trips, taken in non-decreasing order. When we reach a value that is greater than *maximum working time* minutes, *UBdutyLen* is set to the number of trips used in the sum. Initially, we used the number of trips as a rough estimate for *UBnumDut*. As the definitive value for *UBnumDut* we used the number of duties on the first feasible solution found by the CLP solver. Each X_{ij} element represents a single trip and is a finite domain variable with domain $[1..NT]$, where $NT = \mathit{UBdutyLen} \times \mathit{UBnumDut}$. Real trips are numbered from 1 to N, where $N \leq NT$. Trips numbered $N+1$ to NT are *dummy trips*. To simplify the writing of some constraints, the last trip in each line of X is always a dummy trip. A proper choice of the start time, duration and relief points of the dummy trips avoids time and relief point incompatibilities among them and, besides, prevents the occurrence of dummy trips between real trips. Moreover, the choice of start times for all dummy trips guarantees that they occupy consecutive cells at the end of every line in X. The start time of the first dummy trip equals the arrival time of the last real trip plus one minute and its duration is zero minutes. All the subsequent dummy trips also last zero minutes and their start times are such that there is a one minute idle interval between consecutive dummy

Table 6: Pure CLP models, OS 2222 data set

			First Model		Improved Model			
			Feasible		*Feasible*		*Optimal*	
# Trips	# FD	Opt	Sol	Time	Sol	Time	Sol	Time
10	63	7	7	0.35	7	0.19	7	0.63
20	306	11	11	12.21	11	0.47	11	9.22
30	1,032	14	15	00:02:32	15	0.87	14	00:29:17
40	5,191	14	15	00:14:27	15	0.88	-	> 24:00:00
50	18,721	14	15	00:53:59	15	0.97	-	-
60	42,965	14	-	-	15	2.92	-	-
70	104,771	14	-	-	16	3.77	-	-
80	212,442	16	-	-	19	8.66	-	-
90	335,265	18	-	-	24	17.97	-	-
100	496,970	20	-	-	27	29.94	-	-
110	706,519	22	-	-	27	39.80	-	-
125	1,067,406	25	-	-	32	00:01:21	-	-

trips, i.e., they start at each following minute. Their departure and arrival relief points are equal to 0. Using this representation, the set partitioning condition can be easily met with an `alldifferent` constraint applied to a list that contains all the X_{ij} variables.

Five other matrices were used: *Start*, *End*, *Dur*, *DepRP* and *ArrRP*. Cell (i, j) of these matrices represents, respectively, the start time, the end time, the duration, and the departure and arrival relief points of the trip assigned to X_{ij}. Next, we state constraints in the form `element(`$X_{ij}, S, Start_{ij}$`)`, where S is a list containing the start times of all the NT trips. The semantics of this constraint assures that $Start_{ij}$ is the k-th element of list S where k is the value in X_{ij}. This maintains the desired relationship between matrices X and *Start*. Whenever X_{ij} is updated, $Start_{ij}$ is also modified, and vice-versa. Similar constraints are stated between X and each one of the four other matrices. Now, we can write

$$
\begin{aligned}
End_{ij} &\leq Start_{i(j+1)} && (4)\\
ArrRP_{ij} + DepRP_{i(j+1)} &\neq 3 && (5)\\
Idle_{ij} &= BD_{ij} \times \left(Start_{i(j+1)} - End_{ij}\right) && (6)
\end{aligned}
$$

for all $i \in \{1, \ldots, UBnumDut\}$ and all $j \in \{1, \ldots, UBdutyLen-1\}$. Equation (4) guarantees that trips overlapping in time are not in the same duty. Since the maximum number of relief points is two, an incompatibility of two consecutive trips is prevented by (5). In (6), the binary variables BD_{ij} are such that $BD_{ij} = 1$ if and only if $X_{i(j+1)}$ contains a real trip.

Hence, the constraint on total working time, for each duty i, is given by

$$\sum_{j=1}^{UBdutyLen-1} (Dur_{ij} + BI_{ij} \times Idle_{ij}) \leq Workday\ , \tag{7}$$

where BI_{ij} is a binary variable such that $BI_{ij} = 1$ if and only if $Idle_{ij} \leq Idle\_Limit$. The constraint on total rest time is

$$Workday - \sum_{j=1}^{UBdutyLen-1} Dur_{ij} + \sum_{j=1}^{UBdutyLen-1} (Idle_{ij} - BI_{ij} \times Idle_{ij}) \geq Min\_Rest \tag{8}$$

for each duty i. Note that *Idle*, *BD* and *BI* are also matrices in the CLP program. For split-shift duties, we also impose that at most one of the $Idle_{ij}$ variables can assume a value greater than *Idle_Limit*. This is done with an `atmost` constraint in the following manner, for each duty i: `atmost(1,`L`,0)`. If list L contains all the BI_{ij} variables of (7), this means that at most one of them can assume the value zero.

1.5.2 Refinements and Results

The execution time of this model was further improved by:

Elimination of Symmetries — Solutions that are permutations of lines of X are equivalent. To bar such equivalences, the first column of the X matrix was kept sorted. Since exchanging the position of dummy trips gives equivalent solutions, new constraints were used to prevent this from happening when backtracking.

Domain Reduction — Certain trips can only appear on a subset of the available cells. For instance, the first real trip can only appear in $X_{1,1}$.

Use of Another Viewpoint — As in [10], different viewpoints were also used. New Y_k variables were introduced representing "the cell that stores trip k", as opposed to the X_{ij} variables that mean "the trip that is put in cell ij" (an ij cell can be represented by the number $(i-1) \times UBdutyLen + j$). The Y_k variables were connected to the X_{ij} variables through *channeling constraints*. The result is a redundant model with improved propagation properties.

Different Labeling Strategies — Various labeling strategies were tried, including the one developed by [20]. The strategy of choosing the next variable to label as the one with the smallest domain (*first-fail*) was the most effective one. After choosing a variable, it is necessary to select a value from its domain following a specific order, when backtracking occurs. We tested different labeling orders, like increasing, decreasing, and also middle-out and its reverse. Experimentation showed that labeling by increasing order achieved the best results. On the other hand, when using viewpoints, the heuristic developed by [20] rendered the model roughly 15% faster. The basic idea is to label an X variable according to the domain size of the associated Y variables. In our case, for instance, if the current domain of variable $X_{2,5}$ is $[1,7,8]$, the first value to be selected for labeling will be 8 if and only if Y_8 has the smallest domain among variables Y_1, Y_7 and Y_8.

The improved purely declarative model produced feasible schedules in a very good time, as indicated in Table 6, under columns "Improved Model". Obtaining provably optimal solutions, however, was still out of reach for this model. [18] and [12] have also reported difficulties when trying to solve crew scheduling problems with a pure CLP approach. Finding the optimal schedule reduces to choosing, from an extremely large set of elements, a minimal subset that satisfies all the problem constraints. The huge search spaces involved can only be dealt with satisfactorily when pruning is enforced by strong local constraints. Besides, a simple search strategy, lacking good problem specific heuristics, is very unlikely to succeed. When solving scheduling problems of this nature and size to optimality, none of the these requirements can be met easily, rendering it intrinsically difficult for pure CLP techniques to produce satisfactory results in these cases.

The comparative performance of the previous four isolated approaches can be more clearly appreciated through the graph in Fig. 1, which summarizes the results for the OS 2222 data set. The curves are identified as follows: "CLP" is the Constraint Logic Programming approach; "CG+DP" is the column generation approach based on Dynamic Programming; "IP" is the Integer Programming approach and "CFT" is the heuristic approach.

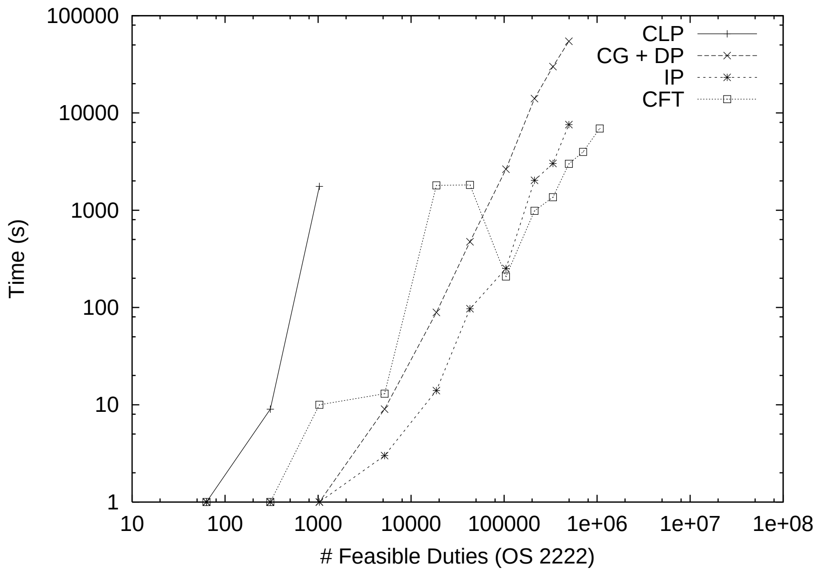


Figure 1: Performance of the isolated approaches over OS 2222

1.6 A Hybrid Approach

[15] has shown that, in some cases, neither the pure IP nor the pure CLP approaches are capable of solving certain kinds of combinatorial problems satisfactorily. But a hybrid

strategy might outperform them.

When contemplating a hybrid strategy, it is necessary to decide which part of the problem will be handled by a constraint solver, and which part will be dealt with in a more classical way. Given the huge number of columns at hand, a column generation algorithm seemed to be almost mandatory. As reported in Sect. 1.4.2, we already knew that the dynamic programming column generator used in the pure IP approach did not perform well. On the other hand, a declarative language is particularly suited to express not only the constraints imposed by the original problem, but also the additional constraints that must be satisfied when looking for feasible duties with negative reduced costs. Given that, it was a natural decision to implement a column generation approach where new columns were generated on demand by a constraint program. Additionally, the discussion in Sect. 1.5.2 indicated that the CLP strategy implemented was very efficient when identifying feasible duties. It lagged behind only when computing a provably optimal solution to the original scheduling problem, due to the minimization constraint. Since it is not necessary to find a column with *the* most negative reduced cost, the behavior of the CLP solver was deemed adequate. It remained to program the CLP solver to find a set of new feasible duties with the extra requirement that their reduced cost should be negative.

There have been other attempts that somehow explore the idea of integrating IP and CLP into column generation algorithms. In the sequel, we identify their main similarities and differences with respect to our approach.

An early work which deals with the cooperation of linear and finite-domain constraint solvers for column generation is [23]. The authors model a bin-packing configuration problem posting constraints both to a linear solver (a revised Simplex algorithm) and to a finite-domain constraint solver. *All* possible bin configurations (columns) are generated at the start and then a pure integer linear problem is solved in order to find the right quantities for each type of bin.

In [21], the authors solve an airline crew assignment problem where the column generation subproblem is modeled as a Constrained Shortest Path Problem (CSPP) on a directed acyclic graph (DAG). This subproblem is formulated as a constraint satisfaction problem. Nevertheless, although they argue that their results are encouraging, the models and computational results are not explicitly described. Moreover, they introduce some heuristic pruning techniques which may prevent the algorithm from finding a provably optimal solution.

[9] describes an iterative cooperation between CLP and linear programming optimizers for solving the pairing generation problem for airline companies. In this case, the generation process is guided by heuristics for choosing "nice" pairings and meta-heuristics which restrict the exploration of the search tree. Also, this algorithm is not a branch-and-price algorithm and the computational experiments are not quite elucidative because of the small number of instances.

[22] present a general framework for column generation based on Constraint Programming. Sometimes, the subproblem of finding new columns with negative reduced cost happens to be too complicated for traditional Operations Research (OR) methods. In these cases, formulating the column generator as a constraint satisfaction problem may help. This is more or less the same idea presented in our previous work [30]. It is interesting to note

here that these two investigations, although leading to similar proposals, have been developed independently and in parallel, and did not borrow ideas from each other. In [22], the framework is instantiated for solving a Crew Assignment problem and the implementation of an efficient path constraint for the subproblem is discussed. Their application does not give rise to the need of integrating this framework inside a branch-and-price algorithm but, according to the authors, this would not be a problem.

Both [14] and [27] make use of the constraint-based column generation framework presented in [22]. In [14], the authors address one kind of Cutting Stock Problem where the column generation subproblem happens to be a Constrained Knapsack Problem (CKP) rather than the usual CSPP. However, the paper concentrates on solving the subproblem efficiently and does not give details about the whole master-slave interaction and the results obtained for the overall Cutting Stock Problem. In [27], the authors describe an algorithm which integrates a Direct Constraint Programming Based Approach (DCPA) and a CP-based Column Generation Approach (CPCGA), in an iterative way, for the crew assignment problem. The pool of columns for the master problem is initialized with a set of initial feasible solutions found by the DCPA. Then, the CPCGA finds a solution for a set covering formulation and the DCPA tries to generate a set partitioning solution through deassignment of variables. Some local refinements on this solution are performed and the CPCGA is called again. They show that, in the long run, this cooperation performs better than both the DCPA or CPCGA alone. However, it is difficult to have a good notion with respect to the effectiveness of their approach since the computational experiments are restricted to two instances. Besides, there is no guarantee of optimality and no idea of the quality of the solutions is presented.

Our hybrid approach differs from the aforementioned approaches due to the following main aspects: we make use of a complete branch-and-price framework, i.e. the linear relaxation of every node of the branch-and-bound tree is solved by means of a Column Generation algorithm; since the total number of feasible columns is enormous, we do not generate them all in advance; the subproblem of column generation is not formulated as a CSPP on a DAG; our experiments are conducted over large real-world data sets; and we guarantee the optimality of the final solutions.

1.6.1 Implementation Issues

The basis of this new algorithm is the same as the one developed for the column generation approach, described in Sect. 1.4.2. The dynamic programming routine is substituted for an ECL^{i}PS^e process that solves the slave subproblem and communicates with the ABACUS process through a network connection. When the ABACUS process has solved the current master problem to optimality, it sends the values of the dual variables to the CLP process. If there remain some columns with negative reduced costs, some of them are captured by the CLP solver and are sent back to the ABACUS process, and the cycle starts over. If there are no such columns, the LP solver has found an optimal solution. Having found the optimal solution for this node of the enumeration tree, its dual bound has also been determined. The normal branch-and-bound algorithm can then proceed until it is time to solve another LP. This interaction is depicted in Fig. 2.

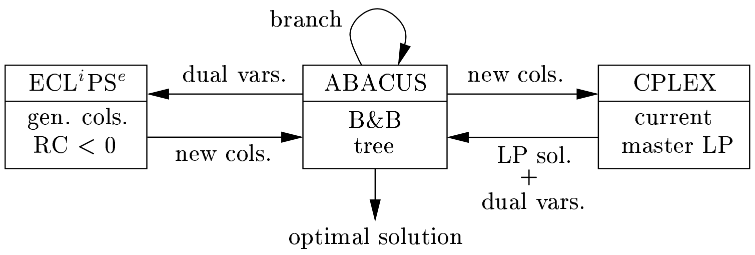


Figure 2: Simplified scheme of the hybrid column generation method

The code for the CLP column generator is almost identical to the code for the improved CLP model, presented in Sect. 1.5.1. There are three major differences. Firstly, the matrix X now has only one row, since we are interested in finding *one* feasible duty and not a complete solution. Secondly, there is an additional constraint stating that the sum of the values of the dual variables associated with the trips in the duty being constructed should represent a negative reduced cost. Using the formula to calculate the reduced cost of a column (feasible duty) given in Section 1.4.2, this constraint reads

$$\sum_{i=1}^{UBdutyLen} C_i > 1\,. \tag{9}$$

For each i, C_i is determined by `element`(X_i, U, C_i), where U is a list whose elements are the values of the dual variables associated with each trip. The dual variables associated with dummy trips are assigned the value zero.

Finally, the minimization predicate was exchanged for a predicate that keeps on looking for new feasible duties until the desired number of feasible duties with negative reduced costs have been computed, or until there are no more feasible assignments. By experimenting with the data sets at hand, we determined that the number of columns with negative reduced cost to request at each iteration of the CLP solver was best set to 53. The redundant modeling, as well as the heuristic suggested by [20], both used to improve the performance of the original CLP formulation, now represented unnecessary overhead and were removed.

1.6.2 Computational Results

The hybrid approach was able to construct an optimal solution to substantially larger instances of the problem, in a reasonable time. Computational results for OS 2222 and OS 3803 appear on Tables 7 and 8, respectively. Column headings # Trips, # FD, Opt, DBR, # CA, # LP and # Nodes stand for, respectively, number of trips, number of feasible duties, optimal solution value, dual bound at the root node, number of columns added, number of linear programming relaxations solved, and number nodes visited. The execution times are divided in three columns: PrT, LPT and TT, meaning, respectively, time spent generating columns, time spent solving linear programming relaxations, and total execution time. In *every* instance, the dual bound at the root node was equal to the value of the

Table 7: Hybrid algorithm, OS 2222 data set (1 relief point)

# Trips	# FD	Opt	DBR	# CA	# LP	# Nodes	PrT	LPT	TT
10	63	7	7	53	2	1	0.08	0.02	0.12
20	306	11	11	159	4	1	0.30	0.04	0.42
30	1,032	14	14	504	11	1	1.48	0.11	2.07
40	5,191	14	14	1,000	26	13	8.03	0.98	9.37
50	18,721	14	14	1,773	52	31	40.97	3.54	45.28
60	42,965	14	14	4,356	107	41	00:04:24	14.45	00:04:40
70	104,771	14	14	2,615	58	7	00:01:36	4.96	00:01:42
80	212,442	16	16	4,081	92	13	00:01:53	18.84	00:02:13
90	335,265	18	18	6,455	141	11	00:02:47	31.88	00:03:22
100	496,970	20	20	8,104	177	13	00:06:38	51.16	00:07:34
110	706,519	22	22	11,864	262	21	00:16:53	00:02:28	00:19:31
125	1,067,406	25	25	11,264	250	17	00:19:09	00:01:41	00:21:00

optimal integer solution. Hence, the LP relaxation of the problem already provided the best possible lower bound on the optimal solution value. Also note that the number of nodes visited by the algorithm was kept small. The same behavior can be observed with respect to the number of columns added.

The sizable gain in performance is shown in the last three columns of each table. Note that the time to solve all linear relaxations of the problem was a small fraction of the total running time, for both data sets.

It is also clear, from Table 7, that the hybrid approach was capable of constructing a provably optimal solution for the smaller data set using 21 minutes of running time on a 350 MHz desktop PC. That problem involved in excess of one million feasible columns and was solved considerably faster when compared with the best performer (see Sect. 1.4.3) among all the previous approaches.

The structural difference between both data sets can be observed by looking at the 100 trip row, in Table 8. The number of feasible duties on this line is, approximately, the same number of one million feasible duties that are present in the totality of 125 trips of the first data set, OS 2222. Yet, the algorithm used roughly twice as much time to construct the optimal solution for the first 100 trips of the larger data set, as it did when taking the 125 trips of the smaller data set. Also, the algorithm lagged behind the heuristic for OS 3803, although the latter was unable to go beyond 110 trips, due to excessive memory usage.

Finally, when we fixed a maximum running time of 24 hours, the algorithm was capable of constructing a solution, and prove its optimality, for as many as 150 trips taken from the larger data set. This corresponds to an excess of 12 million feasible duties. It is noteworthy that less than 60 MB of main memory were needed for this run. A problem instance with as many as $150 \times (12.5 \times 10^6)$ entries would require over 1.8 GB when loaded into main

Table 8: Hybrid algorithm, OS 3803 data set (2 relief points)

# Trips	# FD	Opt	DBR	# CA	# LP	# Nodes	PrT	LPT	TT
20	978	6	6	278	7	1	2.11	0.08	2.24
30	2,890	10	10	852	19	1	9.04	0.20	9.38
40	6,705	13	13	2,190	48	1	28.60	1.03	30.14
50	17,334	14	14	4,220	94	3	00:01:22	3.95	00:01:27
60	45,236	15	15	8,027	175	1	00:03:48	14.81	00:04:06
70	107,337	15	15	11,622	258	1	00:07:42	40.59	00:08:37
80	256,910	15	15	8,553	225	1	00:10:07	47.12	00:10:58
90	591,536	15	15	9,827	269	1	00:14:34	00:02:04	00:16:43
100	1,180,856	15	15	13,330	375	1	00:39:03	00:04:37	00:43:49
110	2,015,334	15	15	13,717	387	1	01:19:55	00:03:12	01:23:19
120	3,225,072	16	16	18,095	543	13	04:02:18	00:09:09	04:11:50
130	5,021,936	17	17	28,345	874	23	06:59:53	00:30:16	07:30:56
140	8,082,482	18	18	27,492	886	25	13:29:51	00:28:56	13:59:40
150	12,697,909	19	19	37,764	1,203	25	21:04:28	00:49:13	21:55:25

memory. By efficiently dealing with a small subset of the feasible duties, our algorithm managed to surpass the memory bottleneck and solve instances that were very large. This observation supports our view that a CLP formulation of column generation was the right approach to solve these very large crew scheduling problems.

The comparative performance of the hybrid model against the isolated IP model over the OS 2222 and OS 3803 data sets is depicted in Figs. 3 and 4, respectively. We chose the IP approach for this comparison for it was the best one among the exact isolated approaches. The curves are identified as follows: "IP" is the Integer Programming approach and "Hybrid" is the hybrid column generation approach.

2 The Crew Rostering Problem

The duties obtained as output from the solution of the crew scheduling phase must be assigned to crews day after day, throughout an entire planning horizon. This sequencing has to obey a set of constraints that differs from the constraints which are relevant to the crew scheduling problem. This set includes, for example, the need for days off, with a certain periodicity, and a minimum rest time between consecutive workdays.

2.1 Input Data

The set of duties to be performed on weekdays is different from the set of duties to be performed on weekends or holidays, due to fluctuations on customer demand. Therefore,

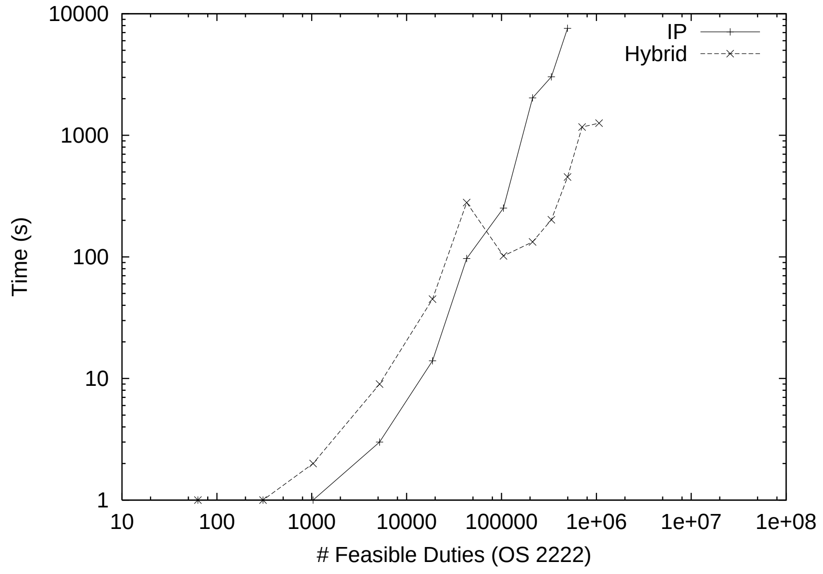


Figure 3: Hybrid column generation vs. Integer Programming over OS 2222

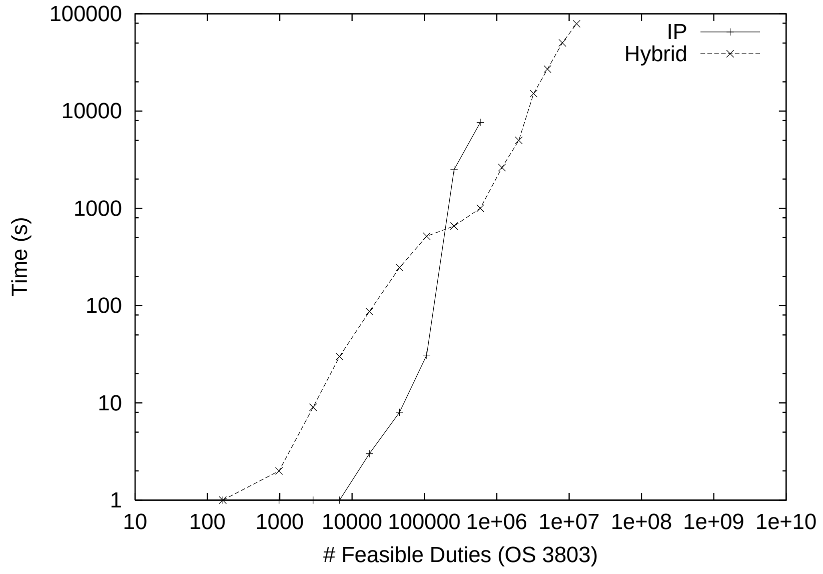


Figure 4: Hybrid column generation vs. Integer Programming over OS 3803

the crew scheduling problem gives as input for the rostering problem a number of distinct sets of duties.

The planning horizon we are interested in spans one complete month. It is important to take into account as input data many features of the month under consideration, such as: the total number of days, which days are holidays and which day of the week is the first day of the month (the remaining weekdays can be easily figured out from this information). The differences in the input data from one month to the next one may lead to variations on the number of crews actually working in each month. Consequently, some rules must be observed in order to select the crews that are going to be effectively used. If, say, in month m 40 crews were needed, and in month $m+1$ only 38 will be necessary, how to select the 2 crews that are going to be left out? Furthermore, suppose that, after eliminating those crews that cannot work on the current month for some reason, the company has 50 crews available. Even if the number of crews remains the same, e.g. 40, from one month to the next one, it is important to evenly distribute the work among them. This balance can be obtained considering the number of days each crew has worked since the beginning of the year, for example, or with the aid of another kind of ranking function for the crews. Finally, since some constraints refer to a time window that spans more than one month (see Sect. 2.2) some attributes, for each crew, have to be carried over between successive months.

The input data needed to build the roster for month m is the following:

- The sets of duties D_{wk}, D_{sa}, D_{su} and D_{ho} which have to be performed on weekdays, Saturdays, Sundays and on holidays, respectively;
- The number of days, d, in month m;
- The occurrence of holidays in month m;
- The day of the week corresponding to the first day in month m;
- The whole set of crews, C, employed by the company;
- For each crew i in C:
  - The set of days, OFF_i, in which i is off duty (e.g. vacations, sickness), excluding its ordinary weekly rests;
  - The number of days, ls_i, between the last Sunday i was off duty and the first day of month m;
  - A binary flag, wr_i, that is equal to 1 if and only if i had a weekly rest in the last week of month $m-1$;
  - A binary flag, sl_i, that is equal to 1 if and only if i performed a split-shift duty during the last week of month $m-1$;
  - The difference in minutes, lw_i, between the last minute i was working in month $m-1$ and the first minute of the first day of month m;
- For each duty k in $D_{\mathrm{wk}} \cup D_{\mathrm{sa}} \cup D_{\mathrm{su}} \cup D_{\mathrm{ho}}$:

  - The start and end times of k (ts_k and te_k, respectively), given in minutes after midnight;
  - A binary flag, ss_k, that equals 1 if and only if k is a split-shift duty;

- A list of all crews in C sorted according to a certain ranking function. This ordering will be used to assign priorities to the crews when identifying the subset of C that is going to work in month m.

2.2 Problem Constraints

The constraints associated to the sequencing of the duties are:

(a) The minimum rest time between consecutive workdays is 11 hours;

(b) Every employee must have at least one day off per week. Moreover, for every time window spanning 7 weeks, at least one of these days off must be on a Sunday;

(c) When an employee performs one or more split-shift duties during a week, his day off in that week must be on Sunday;

(d) In every 24-hour period starting at midnight, within the whole planning horizon, each crew can start to work on at most one duty.

2.3 Objectives

For each month, we are looking for the cheapest solution in terms of the number of crews needed to perform all the duties requested. Additionally, it is desirable to have balanced workloads among all the crews involved, but the models we present in this article are not concerned with this issue yet.

2.4 The Format of the Input Data Sets

Before describing the IP and CLP models for the rostering problem, it is important to understand the format of the instances used in the computational experiments. These instances correspond to the actual schedules constructed by the crew scheduling phase described in Sect. 1. Using the duties built during the crew scheduling phase, we have constructed a set of instances ranging from small ones up to large-sized ones, typically encountered by the management personnel in the bus company. The main features of these instances appear in Table 9.
The *Name* is just a string identifying the instance. The number of crews available for the roster, c, appears under the heading *#Crews*. The column *#Days* shows the number of days in the planning horizon in the format d (h), where d is the total number of days and h indicates how many of those d days are holidays. The next four columns show the number of duties that must be performed in each kind of the possible working days: weekdays, Saturdays, Sundays and holidays, respectively. The format used is ss/tt, where tt is the total number of duties and ss represents how many of the tt duties are split-shift duties. To

Table 9: Description of the instances for the experiments

Name	#Crews	#Days	# Duties Week	Sat	Sun	Holy
string	c	d (h)	$ss_{\text{wk}}/tt_{\text{wk}}$	$ss_{\text{sa}}/tt_{\text{sa}}$	$ss_{\text{su}}/tt_{\text{su}}$	$ss_{\text{ho}}/tt_{\text{ho}}$

begin with, we set the following parameters, for every crew i: $OFF_i = \emptyset$, $ls_i = 0$, $wr_i = 1$, $sl_i = 0$ and $lw_i = 660$. This is the same as ignoring any information from the previous month when constructing the roster for the current month.

2.5 An Integer Programming Approach

Let n be the total number of crews available and let d be the number of days in the current month m. Moreover, let p, q, r and s be the numbers of duties to be performed on weekdays, Saturdays, Sundays and holidays, respectively (i.e. $|D_{\text{wk}}| = p$, $|D_{\text{sa}}| = q$, $|D_{\text{su}}| = r$ and $|D_{\text{ho}}| = s$).

The IP formulation of the rostering problem is based on x_{ijk} binary variables which are equal to 1 if and only if crew i performs duty k on day j. If j is a weekday, k belongs to $\{0, 1, \ldots, p\}$. Analogously, if j is a Saturday, Sunday or holiday, k ranges over $\{0, p+1, \ldots, p+q\}$, $\{0, p+q+1, \ldots, p+q+r\}$ or $\{0, p+q+r+1, \ldots, p+q+r+s\}$, respectively. The duty numbered 0 is a special duty indicating that the crew is off duty on the given day. Thus, if $x_{ij0} = 1$ it means that crew i is not working on day j. For modeling purposes, we set ts_0 to a very large number, $te_0 = 0$ and $ss_0 = 0$.

Given a day j in m, K_j represents its set of duty indexes, except for the duty 0. For instance, if j is a Saturday then $K_j = \{p+1, \ldots, p+q\}$.

2.5.1 The Model

The main objective is to minimize the number of crews working during the present month. This is equivalent to maximizing the number of crews which are idle during the whole month. Let us define new variables $y_i \in \mathbb{R}^+$, for all $i \in \{1, \ldots, n\}$, which are equal to 1 if $x_{ij0} = 1$, for all $j \in \{1, \ldots, d\}$, and are equal to 0 otherwise. To achieve this behavior for the y_i variables, it is necessary to set the objective function as $\max \sum_{i=1}^{n} y_i$ and to impose the following constraints

$$y_i \leq x_{ij0}, \ \ \forall\, i,\ \forall\, j \ . \tag{10}$$

Equations (10) combined with the objective function force a y_i variable to be equal to 1 if and only if crew i is idle during the entire month.

The occurrence of days on which the crews are known to be off duty (e.g. previously assigned vacations) is satisfied by setting

$$x_{ij0} \ = \ 1, \ \ \forall\, i,\ \forall\, j \in OFF_i \ . \tag{11}$$

The subsequent formulas take care of the feasibility of the roster (see Sect. 2.2).

Constraints (a) are dealt with in two steps. Equation (12) takes care of the assignment of duties for the first day in month m. For the other days, assume that a crew i does duty k on day $j-1$. The set $K'_j[k]$ of other duties that cannot be taken by the same crew i on day j because of the 660-minute minimum rest time is given by $\{k' \in K_j \mid ts_{k'} - (te_k - 1440) < 660\}$. Therefore, (13) guarantees the minimum rest time between successive days in month m.

$$\begin{aligned} x_{i1k} &= 0, \ \forall\, i, \ \forall\, k \in K_1 \ \text{ s.t. } \ ts_k + lw_i < 660 \ , && (12) \\ x_{i(j-1)k} + \sum_{k' \in K'_j[k]} x_{ijk'} &\le 1, \ \forall\, i, \ \forall\, j \in \{2, \dots, d\}, \ \forall\, k \in K_{j-1} \ . && (13) \end{aligned}$$

Let us define a *complete week* as seven consecutive days, inside month m, ranging from Monday to Sunday. For every complete week, W, in m, we impose the mandatory day off as

$$\sum_{j \in W} x_{ij0} \ge 1, \ \ \forall\, i \ . \qquad (14)$$

If month m does not start with a complete week, let W' be the set of days in the first week of m up to Sunday. Each crew i with $wr_i = 0$ needs to rest in W' and this is achieved with

$$\sum_{j \in W'} x_{ij0} \ge 1, \ \ \forall\, i \ \text{ s.t. } \ wr_i = 0 \ . \qquad (15)$$

The constraint stating that for each period of time spanning 7 weeks each crew must have at least one day off on Sunday can be described as follows. For each crew i such that $ls_i + d \ge 49$, we construct the set T_i containing the Sundays in the first $(49 - ls_i)$ days of m. Then, we impose

$$\sum_{j \in T_i} x_{ij0} \ge 1, \ \ \forall\, i \ \text{ s.t. } \ ls_i + d \ge 49 \ . \qquad (16)$$

Together, (14) to (16) represent constraints (b).

Suppose that the first day of month m is not Monday and let j^* be the first Sunday in m. To satisfy constraint (c) for each crew i such that $sl_i = 1$, we must state that

$$x_{ij^*0} = 1 \ . \qquad (17)$$

Let S_m be the set of Sundays in m after its sixth day and let P_j be the set of split-shift duties on day j. For these Sundays, we respect constraint (c) with

$$x_{ij0} \ge \sum_{k \in P_{j-r}} x_{i(j-r)k}, \ \ \forall\, i, \ \forall\, j \in S_m, \ \forall\, r \in \{1, \dots, 6\} \ . \qquad (18)$$

Table 10: Computational experiments with the IP model

			# Duties						
Name	#Crews	#Days	Week	Sat	Sun	Holy	LB	Sol	Time
s01	10	10 (1)	00/04	00/01	00/01	00/01	4	6	0.62
s02	10	15 (2)	00/04	00/01	00/01	00/01	4	7	1.50
s03	10	20 (2)	00/04	00/01	00/01	00/01	4	6	2.00
s04	10	25 (2)	00/04	00/01	00/01	00/01	4	6	4.33
s05	10	30 (2)	00/04	00/01	00/01	00/01	4	8	20.91
s06	10	30 (2)	01/04	00/01	00/01	00/01	4	6	9.06
s07	10	30 (2)	02/04	00/01	00/01	00/01	4	6	10.61
s08	10	30 (2)	03/04	00/01	00/01	00/01	4	7	6.81
s09	10	30 (2)	04/04	00/01	00/01	00/01	4	8	9.21
s10	10	30 (2)	04/04	01/01	00/01	00/01	4	7	5.05
s11	10	30 (2)	04/04	01/01	00/01	01/01	4	8	8.35
s12	15	30 (2)	00/04	00/01	00/01	00/01	4	5	8.90

Equation (19) guarantees that each crew is assigned exactly one duty in each day, thus satisfying constraints (d). Additionally, (20) represents the implicit constraint that every duty must be performed in each day, except for the special duty 0.

$$x_{ij0} + \sum_{k \in K_j} x_{ijk} = 1, \ \forall\, i, \forall\, j \ , \tag{19}$$

$$\sum_{i=1}^{n} x_{ijk} = 1, \ \forall\, j, \forall\, k \in K_j \ . \tag{20}$$

2.5.2 Computational Results

The computational results obtained with the IP model are shown in Table 10. The figures under the heading *LB* come from lower bounds on the value of the optimal solution returned by the linear programming relaxation of the IP model. Notice however that the objective function described in Sect. 2.5.1 asks for the maximization of the number of idle crews, which is equivalent to minimizing the number of crews needed to compose the roster. For the purpose of comparison with the CLP model, the values in the *LB* and *Sol* columns of Table 10 represent the number of crews actually working, i.e. the total number of crews available minus the value of the objective function. Finding the optimal solution of the instances in Table 10 turned out to be a very difficult task, despite their relatively small size. Hence, the solution value in column *Sol* corresponds to the first integer solution found by the model, for each instance. The linear relaxations and the integer programs were solved with the CPLEX Solver. Although the computation times for the first integral solution are quite small, the gap between the values of the lower bounds and the feasible solutions is noticeable. Further, these values are still not a good indication of the quality of the model, since we are dealing with very small instances. Yet, when trying to find integer solutions for instances with tens of duties in a workday, this model performed very poorly and no answer could be found within 30 minutes of computation time. Therefore, we decided to experiment with a pure Constraint Logic Programming formulation of the problem.

2.6 A Constraint Logic Programming Approach

Having found difficulties when solving the crew rostering problem with a pure IP model, as described in Sect. 2.5, we decided to try a constraint-based formulation. We used the ECL^{i}PS^e finite domain constraint solver to construct and solve the model.

2.6.1 The Model

Let n, d, p, q, r and s be defined as in Sect. 2.5. The main idea of the CLP model for the rostering problem is to represent the final roster as a two-dimensional matrix, X, where each cell X_{ij} contains the duty performed by crew i on day j, for $i \in \{1, \dots, n\}$ and $j \in \{1, \dots, d\}$.

The X_{ij}'s are finite domain variables whose domains depend on the value of j. As in Sect. 2.5, the duties obtained from the crew scheduling phase are numbered according to their classification as duties for weekdays, Saturdays, Sundays or holidays. In this model, we will not have the concept of a special duty for idleness, as the duty numbered 0 in the IP model. In fact, we will have, for each day, a set of *dummy duties* which tell that a certain crew is off duty.

It is easy to see that the number of crews needed to construct a roster must be at least the maximum number of duties that may be present in any given day of the current month. Thus, we can state that $n \geq \max\{p, q, r, s\}$. Consequently, as the number of X variables for each day j is equal to n, if the domains of these variables were restricted to be the set of duties for day j, some of them would have the same value in the final solution. As we will see later, modeling can be simplified if we avoid this situation and here comes the need for the dummy duties. Let K_j be defined as in Sect. 2.5. Moreover, let the total number of duties be calculated as $tnd = p + q + r + s$. The domains of the X_{ij} variables are then defined as

$$X_{ij} \; :: \; K_j \cup \{tnd + 1, tnd + 2, \dots, tnd + (n - |K_j|)\} \quad \forall\, i,\ \forall\, j \ . \tag{21}$$

If X_{ij} is assigned a duty whose number is greater than tnd, it means that crew i is idle on day j.

Three other sets of variables have to be defined in order to facilitate the representation of the constraints. For all k in $\{1, \dots, tnd\}$, let TS, TE and SS be lists of integers defined as follows: $TS[k] = ts_k$, $TE[k] = te_k - 1440$, $SS[k] = ss_k$. The values of ts, te and ss for the dummy duties are $+\infty$, 0 and 0, respectively. The new variables are called $Start_{ij}$, End_{ij}

and $Split_{ij}$ and relate to the X_{ij} variables through `element` constraints:

$$\begin{aligned}&\texttt{element(}X_{ij}, TS, Start_{ij}\texttt{)} \ ,\\ &\texttt{element(}X_{ij}, TE, End_{ij}\texttt{)} \ ,\\ &\texttt{element(}X_{ij}, SS, Split_{ij}\texttt{)} \ .\end{aligned}$$

Now we can state the constraints (a) through (d) in the ECL^{i}PS^e notation.

Equations (22) and (23) assure that the minimum rest time between consecutive duties is 11 hours. Note the special case for the first day of month m.

$$\begin{aligned} Start_{i1} + lw_i &\geq 660, \ \forall\, i \ , &(22)\\ Start_{ij} - End_{i(j-1)} &\geq 660, \ \forall\, i, \ \forall\, j \in \{2, \dots, d\} \ . &(23)\end{aligned}$$

Similarly to what was defined in Sect. 2.5.1, we use the concept of a complete week, W_i, for each crew i, as a list of variables $[X_{it}, X_{i(t+1)}, \dots, X_{i(t+6)}]$, where t is any Monday and $t+6$ is its subsequent Sunday, both in month m. The need for at least one day off during each week is represented by (24), for complete weeks. Notice that this constraint must be repeated for each complete week W_i associated with every crew i. If $wr_i = 0$ and the first day of m is not Monday, we also need to impose (25), for each crew i and initial week W'_i.

$$\begin{aligned} &\texttt{atmost\_less(}6, W_i, tnd+1\texttt{)} \ , &(24)\\ &\texttt{atmost\_less(}|W'_i| - 1, W'_i, tnd+1\texttt{)} \ . &(25)\end{aligned}$$

In (25), $|W'_i|$ denotes the number of elements in the list W'_i. To state that at most N elements of list L can be smaller than V we use the predicate `atmost_less(`N, L, V`)`. This behavior is achieved with the definitions below

```
flags_less([],_,[]) :- !.
flags_less([X|Y],Val,[B|R]) :- #<(X,Val,B), flags_less(Y,Val,R).
atmost_less(N,L,Val) :- flags_less(L,Val,BF), atmost(N,BF,1).
```

To satisfy constraints (b), there is one condition missing, besides (24) and (25), which assumes at least one day off on Sunday, every seven weeks, for every crew. For each crew i, if $ls_i + d \geq 49$, then

$$\texttt{atmost\_less(}|L_i| - 1, L_i, tnd+1\texttt{)} \ , \qquad (26)$$

where L_i is a list containing the X_{ij}'s associated with the Sundays present in the first $(49 - ls_i)$ days of month m.

Constraints (c) also use the concept of complete weeks, but do not include Sundays. We denote this reduced complete week W^*_i as the list $[Split_{it}, Split_{i(t+1)}, \dots, Split_{i(t+5)}]$. Notice that we now consider the *Split* variables instead of the X variables, as when representing constraints (b).

$$\begin{aligned} Split_{it} + \cdots + Split_{i(t+5)} \ \texttt{\#>}\ 0 \quad \texttt{\#=>} \quad X_{i(t+6)} \ \texttt{\#>}\ tnd, \ \forall\, i, \ \forall\, W^*_i \ , &\quad (27)\\ X_{ik} \quad \texttt{\#>} \quad tnd, \ \forall\, i \ . &\quad (28)\end{aligned}$$

By (27), if one of the $\mathit{Split}_{it}, \ldots, \mathit{Split}_{i(t+5)}$ variables equals 1, then crew i must rest on the next Sunday, which corresponds to $X_{i(t+6)}$. The special case of the first week of m, when the month does not start on Monday and $sl_i = 1$, is dealt with by (28). Here, k stands for the first Sunday of month m.

Our choice of variables already guarantees that each crew starts only one duty per day. But we must also make sure that every duty is assigned to one crew on each day. Because of the dummy duties, this condition can be met easily just by forcing the X_{ij} variables to be pairwise distinct, for each day j:

$$\texttt{alldifferent}([X_{1j}, \ldots, X_{nj}])\texttt{,} \quad \forall\, j \ . \tag{29}$$

Finally, we need to preassign the rest days which are known in advance

$$X_{ij} \ \texttt{\#>} \ \mathit{tnd}, \ \forall\, i, \ \forall\, j \in \mathit{OFF}_i \ . \tag{30}$$

Labeling is done over the X_{ij} variables using the first-fail principle.

Since there are different numbers that represent dummy duties, we can have many symmetric solutions. In other words, two rosters that differ only by the placement of dummy duties constitute the same solution. To avoid this problem and reduce the search space, additional constraints had to be inserted into the CLP program. The idea is the following. For each crew i, if j is the first day in the planning horizon when i does not work, then X_{ij} `#=` $\mathit{tnd} + 1$. In general, we enforce that X_{ij} `#=` $\mathit{tnd} + k$ whenever j is the k-th day on which crew i is off duty.

2.6.2 Computational Results

When compared to the IP model of Sect. 2.5, this model performed much better both in terms of solution quality and computation time. As can be seen in Table 11, it was possible to find feasible solutions for fairly large instances in a few seconds. Again, no minimization predicate was used and the solutions presented here are the first feasible rosters encountered by the model.

Some special cases deserve further consideration. When providing 15 crews to build the rosters for instances s16 and s17, the model could not find a feasible solution even after 10 hours of search. Then, after raising the number of available crews in these instances to 16 (s16a) and 18 (s17a), respectively, solutions were easily found. Another interesting observation arises from instance s19. This instance comes from the solution of a complete real world crew scheduling problem. In this problem, the optimal solution for weekdays contains 25 duties, 22 of which are split shifts. As we did not have access to the input data sets for the other workdays, the sets of duties for Saturdays, Sundays and holidays are subsets of the solution given by the scheduling algorithm for a weekday. Instance s19a is made up of the same duties, except that all of them are artificially considered non-split shifts. Notice that the value of the first solution found is significantly smaller for instance s19a than it is for instance s19. This is an indication of how severe is the influence of the constraints (c) introduced in Sect. 2.2. Moreover, we can see from Table 11 that the values of the solutions grow quickly as the number of split-shift duties increases. With this point

Table 11: Computational experiments with the CLP model

			# Duties						
Name	#Crews	#Days	Week	Sat	Sun	Holy	LB	Sol	Time
s01	10	10 (1)	00/04	00/01	00/01	00/01	4	5	0.08
s02	10	15 (2)	00/04	00/01	00/01	00/01	4	5	0.18
s03	10	20 (2)	00/04	00/01	00/01	00/01	4	5	0.23
s04	10	25 (2)	00/04	00/01	00/01	00/01	4	5	0.36
s05	10	30 (2)	00/04	00/01	00/01	00/01	4	5	0.48
s06	10	30 (2)	01/04	00/01	00/01	00/01	4	5	0.52
s07	10	30 (2)	02/04	00/01	00/01	00/01	4	5	0.50
s08	10	30 (2)	03/04	00/01	00/01	00/01	4	6	0.52
s09	10	30 (2)	04/04	00/01	00/01	00/01	4	7	0.52
s10	10	30 (2)	04/04	01/01	00/01	00/01	4	7	0.52
s11	10	30 (2)	04/04	01/01	00/01	01/01	4	7	0.53
s12	15	30 (2)	00/04	00/01	00/01	00/01	4	5	0.90
s13	15	30 (2)	00/10	00/06	00/05	00/05	10	13	1.22
s13a	15	10 (1)	00/10	00/06	00/05	00/05	10	13	0.28
s14	15	30 (2)	03/10	01/06	00/05	01/05	10	13	1.35
s15	15	30 (2)	03/10	03/06	00/05	03/05	10	15	1.36
s16	15	30 (2)	05/10	03/06	00/05	03/05	10	?	$>$ 10:00:00
s16a	16	30 (2)	05/10	03/06	00/05	03/05	10	16	1.49
s17	15	30 (2)	07/10	03/06	00/05	03/05	10	?	$>$ 10:00:00
s17a	18	30 (2)	07/10	03/06	00/05	03/05	10	18	1.78
s18	30	30 (2)	00/20	00/10	00/10	00/10	20	25	4.96
s18a	30	10 (1)	00/20	00/10	00/10	00/10	20	25	1.09
s19	50	30 (2)	22/25	12/15	12/15	12/15	25	47	14.46
s19a	40	30 (2)	00/25	00/15	00/15	00/15	25	33	9.36
s20	40	30 (2)	06/26	02/15	02/15	02/15	26	34	10.50
s20a	40	7 (1)	06/26	02/15	02/15	02/15	26	34	1.56
s21	36	30 (2)	00/31	00/20	00/20	00/20	31	36	8.30
s21a	36	7 (1)	00/31	00/20	00/20	00/20	31	34	1.29

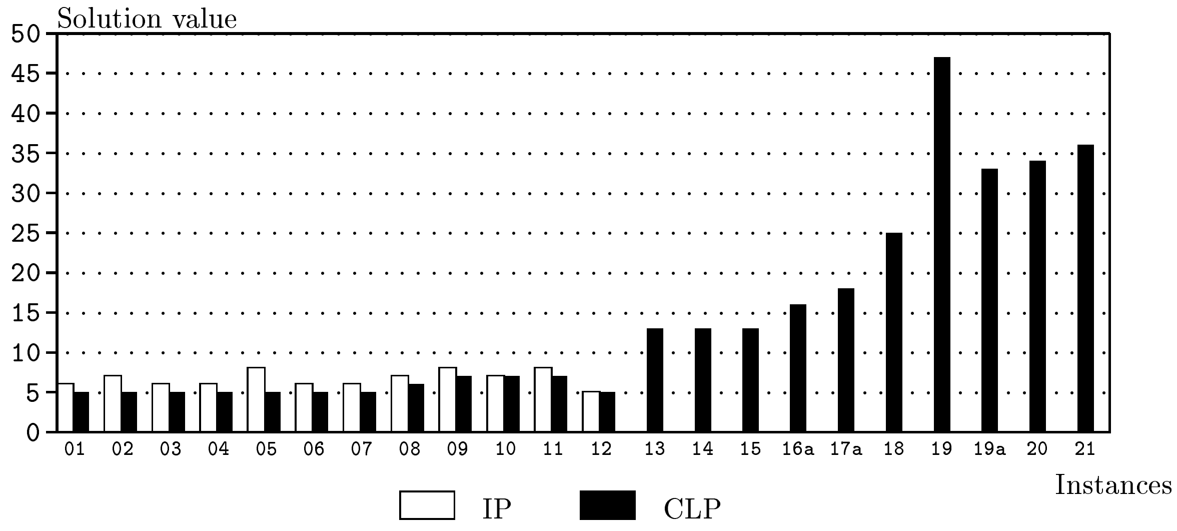


Figure 5: IP vs. CLP in terms of solution quality

in mind, we generated two other solutions for the same crew scheduling problem where the total number of duties used was increased in favor of a smaller number of split shifts. These are s20 and s21. Despite the larger number of duties in the input, the final roster for these instances uses less crews than it did for instance s19. This strengthens the remark made by [6] that, ideally, the scheduling and rostering phases should work cyclicly, with some feedback between them.

All instances in Table 11 do not take into consideration information from the previous month, as mentioned in Sect. 2.4. In order to test the CLP model further, we created one new instance for each instance of Table 11. For these new instances, the values of ls_i, wr_i, sl_i and lw_i, for each crew i, are set taking the feasible solutions of Table 11 as the work profiles of each crew in the preceding month. The behavior of the CLP program was not affected by these more difficult input data sets and we could still find feasible solutions within very short computational times.

Figures 5 and 6 compare the Integer Programming (IP) and Constraint Logic Programming (CLP) models in terms of solution quality and time performance, respectively. The instances on the horizontal axis are named after the same instances from Table 11, except for the letter "s". It is important to remember that, with the IP model, it was only possible to find feasible solutions for instances s01 through s12.

Similarly to the IP approach, this CLP model was not efficient to compute optimal solutions. Being limited to run for 24 hours, we could only find provably optimal solutions for instances s01, s02 and s03.

2.7 Proving Optimality

In Sects. 2.5 and 2.6, we showed that finding provably optimal solutions for this rostering problem is a difficult task. Moreover, it is possible to see from Table 11 that the lower bound provided by the Linear Programming relaxation of the problem is always equal to

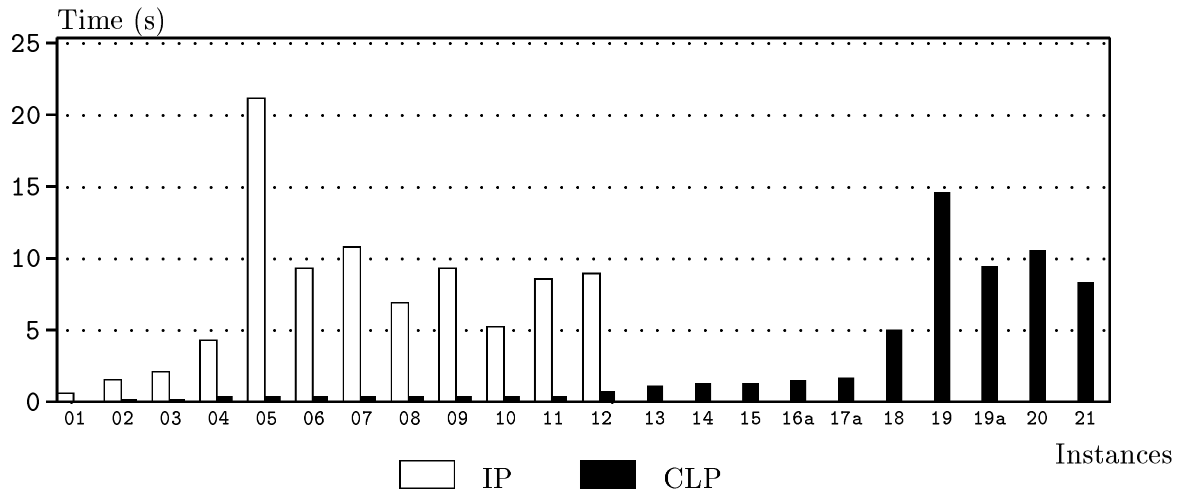


Figure 6: IP vs. CLP in terms of time performance

the largest number of duties that must be performed on a workday. This is clearly a trivial lower bound and probably not a very good one. We decided then to try another formulation for the problem, so as to find better lower bounds or, at least, better feasible solutions.

2.7.1 A Hybrid Model

Another possible mathematical model for the rostering problem turns out to be a typical set partitioning formulation:

$$\begin{aligned} \min \quad & \sum_{j=1}^{n} x_j \\ \text{subject to} \quad & \sum_{j=1}^{n} a_{ij} x_j = 1, \ \ \forall\, i \in \{1, \dots, e\} \\ & x_j \in \{0,1\}, \ \ \forall\, j \in \{1, \dots, n\} \ . \end{aligned}$$

All numbers a_{ij} in the coefficient matrix A are 0 or 1 and its columns are constructed as shown in Fig. 7. Each column is composed of d sequences of numbers, where d is the number of days in the planning horizon. For each $k \in \{1, \dots, d\}$, the k-th sequence, l_k, has length h_k, where h_k is the number of duties that must be performed on day k. Also, at most one number inside each sequence is equal to 1. The number of lines e, in A, equals $\sum_{k=1}^{d} h_k$.

$$(\overbrace{0 \cdots 0 1 0 \cdots 0}^{h_1} \overbrace{0 \cdots 0 1 0 \cdots 0}^{h_2} \cdots \overbrace{0 \cdots 0}^{h_i} \cdots \overbrace{0 \cdots 0 1 0 \cdots 0}^{h_d})^{\mathrm{T}}$$

Figure 7: A column in the coefficient matrix of the set partitioning formulation

Besides having the previous characteristics, a column in A must represent a feasible roster for one crew. More precisely, let $t = (u_1, u_2, \dots, u_d)$ be a feasible roster for a crew, where u_k, $k \in \{1, \dots, d\}$, is the number of the duty performed on day k. Remember from Sect. 2.5.1 that $u_k \in D_k \cup \{0\}$, where D_k is equal to $\{1, \dots, p\}$, $\{p+1, \dots, p+q\}$, $\{p+q+1, \dots, p+q+r\}$ or $\{p+q+r+1, \dots, p+q+r+s\}$, depending on whether k is a weekday, a Saturday, a Sunday or a holiday, respectively. For every such feasible roster t, A will have a column where, in each sequence l_k, the i-th number will be equal to 1 ($i \in \{1, \dots, h_k\}$) if and only if u_k is the i-th duty of D_k. In case $u_k = 0$, all numbers in sequence l_k are set to 0.

With this representation, the objective is to find a subset of the columns of A, with minimum size, such that each line is covered exactly once. This is equivalent to finding a number of feasible rosters which execute all the duties in each day of the planning horizon.

It is not difficult to see that the number of columns in the coefficient matrix is enormous and it is hopeless to try to generate them all in advance. For example, the coefficient matrix for an instance as small as s03 already has billions of columns. Hence, we decided to implement a branch-and-price algorithm to solve this problem, generating columns as they are needed. This approach is considered hybrid because the column generation subproblem is solved by a CLP model. The whole algorithm follows the same basic ideas described in Sect. 1.6. The model for the column generator is a variation of the CLP model of Sect. 2.6. Now, instead of looking for a complete solution for the rostering problem, we are only interested in finding, at each time, feasible rosters corresponding to columns in A with negative reduced costs.

2.7.2 Preliminary Results

The best results for the hybrid model were achieved when setting the initial columns of matrix A as the columns corresponding to the first solution found by the CLP model of Sect. 2.6. Also, the ordinary labeling mechanism worked better than labeling according to the first-fail principle.

With this model, we could find provably optimal solutions for small instances of the rostering problem, as shown in Table 12, where column *Opt* gives the optimal value. By "small instances" we mean either instances with a small number of duties to be executed in each day or instances with a short planning horizon. This is already a noticeable improvement over the pure IP model of Sect. 2.5, which was not able to find any optimal solution, even for the smallest instances. Besides, when comparing Tables 11 and 12, we can see that the first solutions found by the pure CLP model for instances s01 to s04, s13a and s18a are indeed optimal.

We believe that the main reason for the poor performance of this algorithm over larger instances resides on the fact that the IP formulation of Sect. 2.7.1 leads to a highly degenerate problem. When trying to solve larger instances, the pricing subroutine keeps on generating columns indefinitely, with no improvements on the value of the objective function. This is because there are many basic variables with value zero which are replaced by other columns that enter the basis with value zero as well. As a consequence, the linear relaxation of the first node of the branch-and-price enumeration tree could not be completely solved in the medium and large-sized instances. Thus, in order to obtain better linear programming lower bounds, we need to address these degeneracy problems more closely.

Table 12: Computational experiments with the hybrid model

			# Duties					
Name	#Crews	#Days	Week	Sat	Sun	Holy	Opt	Time
s01	10	10 (1)	00/04	00/01	00/01	00/01	5	0.66
s02	10	15 (2)	00/04	00/01	00/01	00/01	5	2.12
s03	10	20 (2)	00/04	00/01	00/01	00/01	5	4.56
s04	10	25 (2)	00/04	00/01	00/01	00/01	5	16.72
s13a	15	10 (1)	00/10	00/06	00/05	00/05	13	12.73
s18a	30	10 (1)	00/20	00/10	00/10	00/10	25	00:04:03
s20a	40	7 (1)	06/26	02/15	02/15	02/15	26	21:23:36
s21a	36	7 (1)	00/31	00/20	00/20	00/20	31	05:39:50

Another problem concerns the labeling policy which follows the simplest possible strategy. In the next section, we present some ideas that were implemented with these deficiencies in mind.

2.7.3 Performance Improvements

We implemented three major modifications in the hybrid algorithm presented so far with the intent to find provably optimal solutions for larger instances of the rostering problem. These modifications are outlined below.

Cost Perturbation. Since the cost of all the columns in our formulation is equal to 1, we have an undesirable symmetry in the sense that any column is, in principle, as suitable for the solution as any other. This fact contributes to intensify the cycling behavior of our highly degenerate model. We decided then to implement one strategy similar to what was presented in [17] and [28]. The basic idea is to add a small perturbation, $\varepsilon \in [-\delta, \delta]$, to the cost of each column. For this mechanism to function correctly, the value of δ may not be chosen arbitrarily. The rule is simple: one solution S with $k+1$ columns must always cost more than one solution S' with k columns. The most critical situation occurs when all columns in S cost $1-\delta$ and all columns in S' cost $1+\delta$. Then, we must have

$$(k+1)(1-\delta) > k(1+\delta)$$

or, equivalently,

$$\delta < \frac{1}{2k+1} \quad . \tag{31}$$

As the number of columns in an optimal solution will never be greater than the total number of lines, e, in the coefficient matrix, we set $k = e$ in (31). One final observation is relevant. If we were solving an integer program with all columns loaded in memory, the value of ε, for each column, could be randomly chosen inside the interval $[-\delta, \delta]$. However, as we are generating columns on demand and the negative reduced cost constraint depends on the cost of the column in the objective function, the choice of ε must be deterministic. Our approach was to divide the $[-\delta, \delta]$ interval into p discrete values and then use a mod-type hash function to map each column to a specific value of perturbation ε. [11] suggest that p should be a prime number not too close to a power of 2. We decided then to set $p = 1531$.

Set Covering Formulation. With the problem constraints described in Sect. 2.2, it is easy to see that any sub-roster of a feasible roster is itself another feasible roster. Hence, if we change the set partitioning formulation of Sect. 2.7.1 to a set covering formulation, the final covering solution can be transformed in a partition just by removing from some rosters those duties that are performed more than once, if any. This idea was motivated by the fact that, in general, a set covering formulation of a problem is easier to solve than a set partitioning formulation for the same problem.

New Labeling Criterion. Recall from Sect. 1.4.2 that the reduced cost constraint for column c reads

$$\sum_{j \in D_c} u_j > \mathrm{Cost}_c \ , \tag{32}$$

where D_c is the set of duties covered by c, u_j is the value of the dual variable associated to duty j and Cost_c is the coefficient of c in the objective function. Following a greedy criterion, we decided to label the variables in the CLP column generator taking into account their contribution to the left hand side of (32). In other words, after choosing one variable to label next, the values in its domain are initially sorted according to the non-increasing order of their corresponding u_j values. That is, the duties with the largest corresponding dual values are tried first. As the sum of u_j's must be greater than Cost_c, if the largest u_j values are not large enough, then there is no need to test the smallest values.

2.7.4 Computational Results with the Improved Algorithm

The inclusion or exclusion of each one of the previous three suggested improvements, lead to eight possible versions of the hybrid algorithm. After comparing the results obtained with all these possible combinations, the best overall performance was achieved by an algorithm using the simplest labeling strategy over a set covering formulation without perturbations on the costs. These results are summarized in Table 13. On the other hand, when tackling the specific instance s20a, the best overall performance was achieved by an algorithm using the improved labeling strategy over a set partitioning formulation without cost perturbation. The latter configuration could find an optimal solution for instance s20a in less than 16 minutes, whereas Table 13 reports more than 12 hours of computation for the same instance.

When comparing Tables 12 and 13, we notice significant gains both in terms of the time needed to find the optimal solutions and in terms of the sizes of the instances that were optimally solved by the algorithm. The improved versions of the hybrid algorithm still do

Table 13: Computational results with the best configuration of the improved hybrid model

Name	#Crews	#Days	# Duties Week	Sat	Sun	Holy	Opt	Time
s01	10	10 (1)	00/04	00/01	00/01	00/01	5	0.31
s02	10	15 (2)	00/04	00/01	00/01	00/01	5	0.47
s03	10	20 (2)	00/04	00/01	00/01	00/01	5	0.62
s04	10	25 (2)	00/04	00/01	00/01	00/01	5	0.73
s05	10	30 (2)	00/04	00/01	00/01	00/01	5	0.85
s06	10	30 (2)	01/04	00/01	00/01	00/01	5	0.89
s07	10	30 (2)	02/04	00/01	00/01	00/01	5	0.87
s13a	15	10 (1)	00/10	00/06	00/05	00/05	13	7.34
s18a	30	10 (1)	00/20	00/10	00/10	00/10	25	20.05
s20a	40	7 (1)	06/26	02/15	02/15	02/15	26	12:40:42
s21a	36	7 (1)	00/31	00/20	00/20	00/20	31	00:17:19

not scale up to an entire planning horizon of one complete month with a large number of duties in each day. Nevertheless, we were able to construct optimal weekly rosters for real world instances. We believe that further developments on the labeling strategy through the inclusion of more sophisticated guiding heuristics can be used to improve the performance of this algorithm.

3 Conclusions and Future Work

Real world crew management problems often give rise to large set covering or set partitioning formulations. We have given a detailed description of urban transit crew management problems that are part of the daily operation of a medium-sized Brazilian bus company. In particular, their rostering problem is rather different from some other bus crew rostering problems found in the literature.

We have shown a way to integrate pure IP and declarative CLP techniques into hybrid column generation algorithms that solved, to optimality, huge instances of these real world crew management problems. Obtaining provably optimal solutions for these problems was a very difficult task for both IP and CLP approaches when taken in isolation. Our hybrid methodology combines the strengths of both sides, while getting over their main weaknesses.

Another crucial advantage of our hybrid approach over a number of previous attempts is that it considers *all* feasible duties. Therefore, the need does not arise to use specific rules to select, at the start, a subset of "good" feasible duties (or rosters). This kind of preprocessing could prevent the optimal solution from being found. Instead, our algorithm implicitly looks at the set of all feasible duties (rosters), when activating the column generation method. When declarative constraint satisfaction formulations are applied to generate new columns on demand, they have shown to be a very efficient strategy, in contrast to Dynamic

Programming, for example.

We believe that our CLP formulations can be further improved. In particular, the search strategy deserves more attention. Earlier identification of unpromising branches in the search tree can reduce the number of backtracks and lead to substantial savings in computational time. Techniques such as dynamic backtracking ([16]) and the use of *nogoods* ([24]) can be applied to traverse the search tree more efficiently, thereby avoiding useless work.

Finally, we would like to thank the Pioneira Bus Company from the city of Belo Horizonte, in Brazil, for kindly providing us with the real world experimental data.

References

[1] 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. Technical Report COC-9403, Georgia Institute of Technology, Atlanta, USA, 1993.

[2] J. E. Beasley and N. Christofides. An algorithm for the resource constrained shortest path problem. *Networks*, 19:379–394, 1989.

[3] L. Bianco, M. Bielli, A. Mingozzi, S. Ricciardelli, and M. Spandoni. A heuristic procedure for the crew rostering problem. *European Journal of Operational Research*, 58(2):272–283, 1992.

[4] A. Caprara, M. Fischetti, and P. Toth. A heuristic method for the set covering problem. Technical Report OR-95-8, DEIS, Università di Bologna, 1995. Published in *Operations Research* 47, 1999.

[5] A. Caprara, M. Fischetti, P. Toth, and D. Vigo. Modeling and solving the crew rostering problem. Technical Report OR-95-6, DEIS, University of Bologna, Italy, 1995. Published in *Operations Research* number 46, 1999.

[6] A. Caprara, M. Fischetti, P. Toth, D. Vigo, and P. L. Guida. Algorithms for railway crew management. *Mathematical Programming*, 79:125–141, 1997.

[7] 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. Technical Report OR-96-12, DEIS, University of Bologna, Italy, 1996. Published in *Software Practice and Experience* number 28, 1998.

[8] P. Carraresi and G. Gallo. A multi-level bottleneck assignment approach to the bus drivers' rostering problem. *European Journal of Operational Research*, 16(2):163–173, 1984.

[9] A. Chabrier. A cooperative CP and LP optimizer approach for the pairing generation problem. In *International Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CP-AI-OR'99)*, Ferrara, Italy, February 1999.

[10] B. M. W. Cheng, K. M. F. Choi, J. H. M. Lee, and J. C. K. Wu. Increasing constraint propagation by redundant modeling: an experience report. *Constraints*, 4(2), May 1999.

[11] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* The MIT Press, 1990.

[12] K. Darby-Dowman and J. Little. Properties of some combinatorial optimization problems and their effect on the performance of integer programming and constraint logic programming. *INFORMS Journal on Computing*, 10(3), 1998.

[13] M. Desrochers and F. Soumis. A column generation approach to the urban transit crew scheduling problem. *Transportation Science*, 23(1), 1989.

[14] T. Fahle and M. Sellmann. Constraint programming based column generation with knapsack subproblems. In *Second International Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CP-AI-OR'00)*, Paderborn, Germany, March 2000.

[15] C. Gervet. Large Combinatorial Optimization Problems: a Methodology for Hybrid Models and Solutions. In *Journées Francophones de Programmation en Logique et par Contraintes*, 1998.

[16] M. L. Ginsberg. Dynamic backtracking. *Journal of Artificial Intelligence Research*, (1):25–46, 1993.

[17] M. Grötschel, A. Martin, and R. Weismantel. Packing steiner trees: A cutting plane algorithm and computational results. *Mathematical Programming*, 72:125–145, 1996.

[18] N. Guerinik and M. Van Caneghem. Solving crew scheduling problems by constraint programming. In *Lecture Notes in Computer Science,* vol. 976, pages 481–498, 1995. Proceedings of the First International Conference on the Principles and Practice of Constraint Programming, CP'95.

[19] J. K. Jachnik. Attendance and rostering system. In A. Wren, editor, *Computer Scheduling of Public Transport*, pages 337–343. North-Holland Publishing Co., 1981.

[20] J. Jourdan. *Concurrent Constraint Multiple Models in CLP and CC Languages: Toward a Programming Methodology by Modeling.* PhD thesis, Université Denis Diderot, Paris VII, February 1995.

[21] U. Junker, S. Karisch, N. Kohl, and B. Vaaben. Constraint programming based column generation for crew assignment. In *International Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CP-AI-OR'99)*, Ferrara, Italy, February 1999.

[22] U. Junker, S. Karisch, N. Kohl, B. Vaaben, T. Fahle, and M. Sellmann. A framework for constraint programming based column generation. In *Lecture Notes in Computer*

Science, vol. 1713, pages 261–274, Alexandria, VA, USA, October 1999. Proceedings of the Fifth International Conference on Principles and Practice of Constraint Programming (CP'99).

[23] M. Leconte, P. Couronne, and D. Vergamini. Column generation using cooperating constraint-based solvers. In *Workshop on Constraints and Constraint Programming*, Singapore, December 1996. Post-Conference Workshop held at the Asian Computing Science Conference (Asian'96).

[24] J. Lever, M. Wallace, and B. Richards. Constraint logic programming for scheduling and planning. *British Telecom Technical Journal*, (13):73–81, 1995.

[25] C. R. Reeves, editor. *Modern Heuristic Techniques for Combinatorial Problems.* Wiley, 1993.

[26] D. M. Ryan and B. A. Foster. An integer programming approach to scheduling. In A. Wren, editor, *Computer Scheduling of Public Transport.* North-Holland Publishing Company, 1981.

[27] M. Sellmann, K. Zervoudakis, P. Stamatopoulos, and T. Fahle. Integrating direct CP search and CP-based column generation for the airline crew assignment problem. In *Second International Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CP-AI-OR'00)*, Paderborn, Germany, March 2000.

[28] E. Uchôa and M. Poggi de Aragão. Vertex-disjoint packing of two steiner trees: Polyhedra and branch-and-cut. In *Lecture Notes in Computer Science,* vol. 1610, pages 439–452, Graz, Austria, 1999. Proceedings of the 7th International Conference on Integer Programming and Combinatorial Optimization (IPCO'99).

[29] F. Vanderbeck. *Decomposition and Column Generation for Integer Programming.* PhD thesis, Université Catholique de Louvain, CORE, September 1994.

[30] T. H. Yunes, A. V. Moura, and C. C. de Souza. A hybrid approach for solving large scale crew scheduling problems. In *Lecture Notes in Computer Science,* vol. 1753, pages 293–307, Boston, MA, USA, January 2000. Proceedings of the Second International Workshop on Practical Aspects of Declarative Languages (PADL'00).