TRANSPORTATION SCIENCE
Vol. 39, No. 2, May 2005, pp. 273–288
ISSN 0041-1655 | EISSN 1526-5447 | 05 | 3902 | 0273

DOI 10.1287/trsc.1030.0078
© 2005 INFORMS

Hybrid Column Generation Approaches for Urban Transit Crew Management Problems

Tallys H. Yunes, Arnaldo V. Moura, Cid C. de Souza
Institute of Computing, University of Campinas, Caixa Postal 6176, CEP 13084-971, Campinas, SP, Brazil
{tallys@ic.unicamp.br, arnaldo@ic.unicamp.br, cid@ic.unicamp.br}

This article considers the overall crew management problem arising from the daily operation of an urban transit bus company that serves the metropolitan area of the city of Belo Horizonte, Brazil. Due to its intrinsic complexity, the problem is divided in two distinct subproblems: *crew scheduling* and *crew rostering*. We have investigated each of these problems using mathematical programming (MP) and constraint logic programming (CLP) approaches. In addition, we 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, they proved to be 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; 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 desktop PC.

Key words: public transportation; crew scheduling; constraint programming; hybrid algorithms; column generation
History: Received: August 2000; revisions received: January 2002, February 2003; accepted: February 2003.

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 (Caprara et al. 1997). 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 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.

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. Because employee wages may account for 50% 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. Because 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 Desrochers and Soumis (1989), 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 toward solving large instances.

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. A good heuristic to improve the search strategy does not come easily, as noted in Darby-Dowman and Little (1998).

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 cost. This model 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 specific union regulations and operational constraints make our rostering problem fairly distinct from other known crew rostering problems found in the literature (such as Caprara et al. 1998; Caprara et al. 1998). 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 to have every duty performed in each day of the planning horizon. In other common cases such as Jachnik (1981), Carraresi and Gallo (1984), and Bianco et al. (1992), 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 the crew in the previous period. Moreover, different crews have different needs for days off, imposed by personal requirements.

Similar to the crew scheduling problem, we started with models based on pure IP and CLP techniques to solve the rostering problem. 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 and includes a number of subsections. In §1.4, we describe an IP approach and report on the implementation of an alternative technique using standard column generation. In §1.5 we consider a pure CLP approach, and in §1.6 we present the hybrid approach. 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 datasets used in our experiments. In §2.5, we briefly comment on an IP formulation used to solve the problem. Section 2.6 presents some experiments that were conducted to evaluate the performance of a pure CLP model for crew rostering. The results achieved with a hybrid column generation approach are analyzed in §2.7. Finally, we draw the main conclusions and discuss further issues in §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 a second. 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 that is the source of the input data. 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 time of day of the trip start. 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, to which each crew is entitled. 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 come 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 daily number of trips and the number of relief points in these instances are representative of the lines in the transportation system of Belo Horizonte, which is composed of 266 different bus lines. The input data tables for these lines are called OS 2222 and OS 3803, respectively. Table 1 shows the first 10 rows of OS 3803. By considering initial segments taken from these two input data 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 Figure 1. This figure was constructed as follows. For each (x, y) entry, the ordinate y indicates how many trips are active at time x. We could also have built smaller instances that span the entire day by selecting trips from the whole instances in a sparser way. We decided not to do this because sparser instances become easier to solve and are not representative of the real situation.

Table 1 Sample from OS 3803

Start time	Time duration	Initial relief point	Final relief point
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

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

$$\textit{total working time} \leq \textit{Workday}$$
$$\textit{rest time} \geq \textit{Min\_Rest}.$$

If trip i precedes trip j in the same duty, we must have

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

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

Restrictions from the operational environment impose

$$\textit{Idle\_Limit} = 120,$$
$$\textit{Workday} = 440,$$
$$\textit{Min\_Rest} = 30,$$

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. Because 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

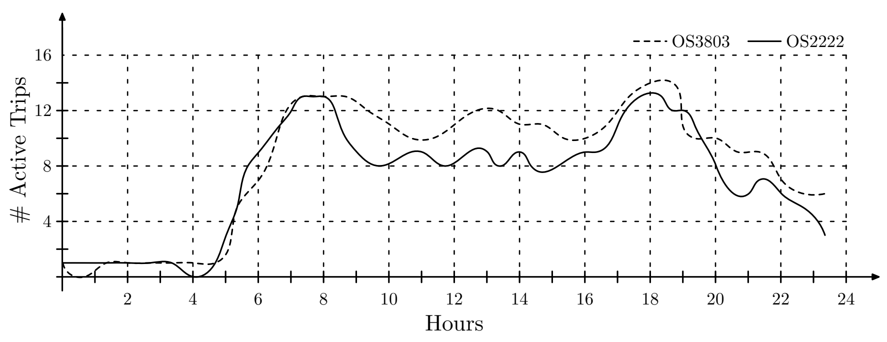

Figure 1 Distribution of Trips Along the Day

problem is

$$\min \sum_{j=1}^{n} x_j \qquad (1)$$

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

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

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 (FD—column "# FD") is also presented. For OS 2222, we get in excess of 1 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 "≥" 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 (ILOG 1999) input format. 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

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

Table 3 Computational Results for OS 2222 (1 Relief Point)

			Pure IP		CG + DP		
# Trips	# FD	Opt	Sol	Time	Sol	Pricing time	Total time
10	63	7	7	0.02	7	0.00	0.01
20	306	11	11	0.03	11	0.04	0.07
30	1,032	14	14	0.06	14	0.43	0.52
40	5,191	14	14	3.04	14	8.82	9.10
50	18,721	14	14	14.29	14	00:01.26	00:01:29
60	42,965	14	14	00:01:37	14	00:07:45	00:07:54
70	104,771	14	14	00:04:12	14	00:43:58	00:44:19
80	212,442	16	16	00:33:52	16	03:53:06	03:53:58
90	335,265	18	18	00:50:28	18	08:18:11	08:18:53
100	496,970	20	20	02:06:32	20	15:07:22	15:08:55
110	706,519	22	—	(*out of memory*)	—	—	(*out of time*)
125	1,067,406	25	—	(*out of memory*)	—	—	(*out of time*)

instance. Besides, memory usage was also increasing at an alarming pace, and execution time was lagging when compared with other approaches that were being developed in parallel. As an alternative, we decided to implement a column generation approach.

1.4.2. Column Generation with Dynamic Programming. Column generation is a technique that is widely used to handle linear programs that have a very large number of columns in the coefficient matrix (see Barnhart et al. 1998). 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 2, looking for columns with negative reduced cost. 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 1. 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 cost 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*.

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, Desrochers and Soumis (1989) suggest reducing the slave subproblem to a constrained shortest path problem (CSPP). Since this technique is still considered to be one of the most efficient (see Crainic and Laporte 1998; Dror 2000), we decided to apply it to our problem.

Implementation and Results. To implement the branch-and-price strategy, the use of the ABACUS (OREAS 1999) branch-and-price framework saved a lot of programming time. We refer the reader interested in more details to Yunes et al. (2000). In Table 3, columns under "CG + DP" show the computational results for OS 2222. Within the maximum allowed Central Processing Unit time of 24 hours, this approach did not reach a satisfactory performance. Because the constrained shortest-path subproblem is solved by means of a pseudopolynomial 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 (pricing time) was responsible for more than 90% of the total execution time, on average, over all instances (see Table 3).

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 CLP techniques. We were able to find feasible schedules in a very short time using the CLP system ECL^{i}PS^e (IC-Parc 2001), but obtaining provably optimal solutions was out of reach for our CLP approach. The details can be found in Yunes et al. (2000a). Guerinik and Caneghem (1995) and Darby-Dowman and Little (1998) also report 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 unlikely to succeed. When solving scheduling problems of this nature and size to optimality, none of these requirements can be met easily, rendering it intrinsically difficult for pure CLP techniques to produce satisfactory results in these cases.

1.6. A Hybrid Approach

Gervet (1998) shows 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. On the one hand, as reported in §1.4.2, we knew that the dynamic programming column generator used in the pure IP approach did not perform well on the largest instances. 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 cost. Given that, it was a natural decision to implement a column generation approach where new columns were generated on demand by a constraint program. In addition, the discussion in §1.5 indicates 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. Because 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. We identify their main similarities and differences with respect to our approach in the following paragraphs.

An early work treating the cooperation of linear and finite-domain constraint solvers for column generation is Leconte et al. (1996). A bin-packing configuration problem is modeled, 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 a pure integer linear model is solved in order to find the right quantities for each type of bin.

In Junker et al. (1999a), the authors solve an airline crew assignment problem where the column generation subproblem is modeled as a 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 that may prevent the algorithm from finding a provably optimal solution.

Chabrier (1999) 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 metaheuristics that 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.

Junker et al. (1999b) present a general framework for column generation based on constraint programming (CP). Sometimes the subproblem of finding new columns with negative reduced cost is too complicated for traditional operations research (OR) methods. In these cases, formulating the column generator as a constraint satisfaction problem might help. This is more or less the same idea presented in our previous work (Yunes et al. 1999). It is interesting to note here that these two investigations, although leading to similar proposals, have been developed independently and in parallel. Junker et al. (1999b), instantiate the framework for solving a crew assignment problem and discuss the implementation of an efficient path constraint for the subproblem. Their application does not give rise to the need of integrating this framework inside a branch-and-price algorithm but, according to them, this would not be a problem.

Both Fahle and Sellmann (2000) and Sellmann et al. (2000) make use of the constraint-based column generation framework presented in Junker et al. (1999b). Fahle and Sellmann (2000) address one kind of cutting stock problem where the column generation subproblem is 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. Sellmann et al. (2000) describe an algorithm that 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. The CPCGA then 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. In addition, there is no guarantee of optimality and no presentation of the idea of the quality of the solutions.

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 datasets; 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 §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 columns with negative reduced cost, 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 linear programming (LP) solver has found an optimal solution. Having found the optimal solution for this node of the enumeration tree, its dual bound is also available. The normal branch-and-bound algorithm can then proceed until it is time to solve another program. This interaction is depicted in Figure 2. For branching purposes, we selected the active node with best LP bound and we branched on a variable with fractional value closest to one half.

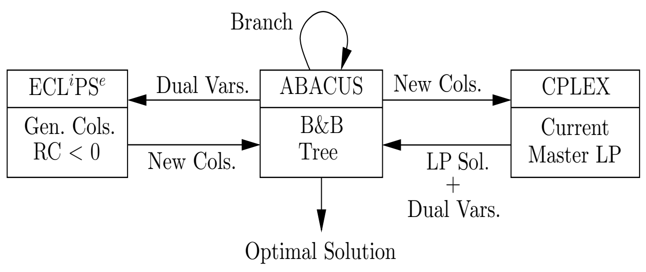

Figure 2 Simplified Scheme of the Hybrid Column Generation Method

The code for the CLP column generator is almost identical to the code for our pure CLP model referred to in §1.5. That is, we use the same representation of the problem constraints. However, there are three major differences. First, we are not looking for a complete schedule any more—just feasible duties. Second, there are two additional kinds of constraints. One states 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. The other states that variables that have been fixed to zero by previous branching decisions must not be generated again. This is done by sending the list of branching variables to the column generator. Finally, we no longer need to use a minimization predicate. Instead, we use a predicate that keeps on looking for new feasible duties until the desired number of feasible duties with negative reduced cost have been computed, or until there are no more feasible assignments. By experimenting with the datasets at hand, we determined that the maximum number of columns with negative reduced cost to request at each iteration of the CLP solver was best set to 50. Redundant modeling techniques (Caseau and Koppstein 1992; Cheng et al. 1999), as well as more advanced labeling heuristics (Jourdan 1995), both used to improve the performance of the original CLP formulation, became unnecessary and were removed. This model is thoroughly explained in Yunes et al. (2000b).

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 in Tables 4 and 5, 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 of nodes visited. Column headings "PrT," "LPT," "HT," and "TT" stand for, respectively, time spent generating columns, time spent solving linear programming relaxations, time spent by a primal heuristic, and total execution time. In every instance the dual bound at the root node was equal to the value of the optimal integer solution. Hence, the LP relaxation of the problem already provided the best possible lower bound on the optimal solution value. This motivated the use of a primal heuristic after the solution of each linear relaxation of the problem. For this purpose, we implemented the set covering heuristic developed by Caprara et al. (1999). 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

Table 4 **Hybrid Algorithm, OS 2222 Dataset (1 Relief Point)**

# Trips	# FD	Opt	DBR	# CA	# LP	# Nodes	PrT	LPT	HT	TT
10	63	7	7	53	2	1	0.08	0.02	0.00	0.12
20	306	11	11	159	4	1	0.30	0.04	0.00	0.42
30	1,032	14	14	504	11	1	1.48	0.11	0.00	2.07
40	5,191	14	14	901	18	1	4.75	0.08	0.00	4.83
50	18,721	14	14	1,007	20	1	6.35	0.12	0.02	6.49
60	42,965	14	14	1,802	35	1	46.38	0.41	0.02	46.81
70	104,771	14	14	2,862	55	1	00:01:12	1.02	4.80	00:01:18
80	212,442	16	16	3,260	69	1	57.00	1.97	5.38	00:01:04
90	335,265	18	18	6,360	121	1	00:02:21	8.45	00:01:41	00:04:10
100	496,970	20	20	7,632	148	7	00:04:40	21.42	00:04:45	00:09:46
110	706,519	22	22	10,494	200	3	00:08:40	26.55	00:02:52	00:11:59
125	1,067,406	25	25	13,303	258	11	00:17:05	00:01:03	00:01:01	00:19:09

crew scheduling. Using our own experience and additional ideas from the chapter on Lagrangian Relaxation in Reeves (1993), an implementation was written in C 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. Although this is a set covering heuristic, it was able to find, in many cases, coverings that were indeed partitions. In those cases that it did not find partitions, we implemented a simple local search algorithm that would try to transform the covering into a partition of the same size by removing trips whenever possible.

For the purpose of comparison, when we solved instances with two relief points without using this primal heuristic within the same time limit of 24 hours, we were only able to find optimal solutions for instances with up to 150 trips. Moreover, in this case the search tree ended up having 25 nodes and the time needed to prove optimality was about 22 hours. As we can see in Table 5, using the primal heuristic we were able to prove optimality for instances of up to 210 trips in approximately 14.5 hours. Notice that this 210-trip instance has roughly five times as many columns as the 150-trip instance. When solving instances with one relief point the heuristic helped us drop the maximum number of nodes in the enumeration tree from 41 to 11. For the instances with two relief points, the maximum number of nodes dropped from 25 to 7.

Finally, note that the number of columns added by the algorithm was kept small. The sizable gain in performance is shown in the last four 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 datasets.

It is also clear, from Table 4, that the hybrid approach was capable of constructing a provably optimal solution for the smaller dataset using 19 minutes of running time on a 350 MHz desktop. That problem involved in excess of one million feasible columns and was solved considerably faster when compared with

Table 5 **Hybrid Algorithm, OS 3803 Dataset (2 Relief Points)**

# Trips	# FD	Opt	DBR	# CA	# LP	# Nodes	PrT	LPT	HT	TT
20	978	6	6	278	7	1	2.11	0.08	0.00	2.24
30	2,890	10	10	852	19	1	9.04	0.20	0.00	9.38
40	6,705	13	13	2,190	48	1	28.60	1.03	0.00	30.14
50	17,334	14	14	4,187	80	1	00:01:03	1.27	0.00	00:01:04
60	45,236	15	15	8,027	175	1	00:03:48	14.81	0.00	00:04:06
70	107,337	15	15	11,622	258	1	00:07:42	40.59	0.00	00:08:37
80	256,910	15	15	8,553	225	1	00:10:07	47.12	0.00	00:10:58
90	591,536	15	15	9,827	269	1	00:14:34	00:02:04	0.00	00:16:43
100	1,180,856	15	15	13,330	375	1	00:39:03	00:04:37	0.00	00:43:49
110	2,015,334	15	15	13,717	387	1	01:19:55	00:03:12	0.00	01:23:19
120	3,225,072	16	16	20,011	379	1	02:01:07	00:02:35	00:01:14	02:04:56
130	5,021,936	17	17	10,303	199	7	01:00:07	57.31	00:01:31	01:02:35
140	8,082,482	18	18	8,703	303	7	02:06:40	00:01:18	57.92	02:08:56
150	12,697,909	19	19	9,487	182	5	01:54:09	00:01:25	00:01:19	01:56:53
200	54,365,975	25	25	22,154	419	1	14:06:45	00:08:01	00:06:21	14:21:07
210	67,756,512	26	26	23,691	451	7	14:13:49	00:10:41	00:15:08	14:39:38

the best performer (see §1.4.1) among all the previous approaches.

The structural difference between both datasets can be observed by looking at the 100-trip row, in Table 5. The number of feasible duties on this line is, approximately, the same number of 1 million feasible duties that are present in the totality of 125 trips of the first dataset, OS 2222. Yet the algorithm used roughly twice as much time to construct the optimal solution for the first 100 trips of the larger dataset as it did when taking the 125 trips of the smaller dataset. In essence, the instances with two relief points are harder to solve than those with one relief point because they involve more trips per time unit. In other words, they are denser, as can be seen in Figure 1 and Table 2.

Finally, when we fixed a maximum running time of 24 hours, the algorithm was capable of constructing a solution and proving its optimality for as many as 210 trips taken from the larger dataset. This corresponds to an excess of 67 million feasible duties. It is noteworthy that less than 30 MB of main memory were needed for this run. 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 datasets is depicted in Figures 3 and 4, respectively. We chose the IP approach for this comparison because 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.

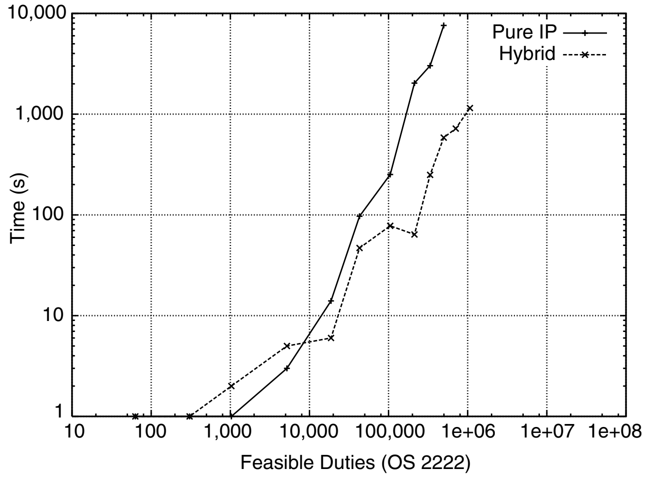

Figure 3 Hybrid Column Generation vs. Integer Programming Over OS 2222

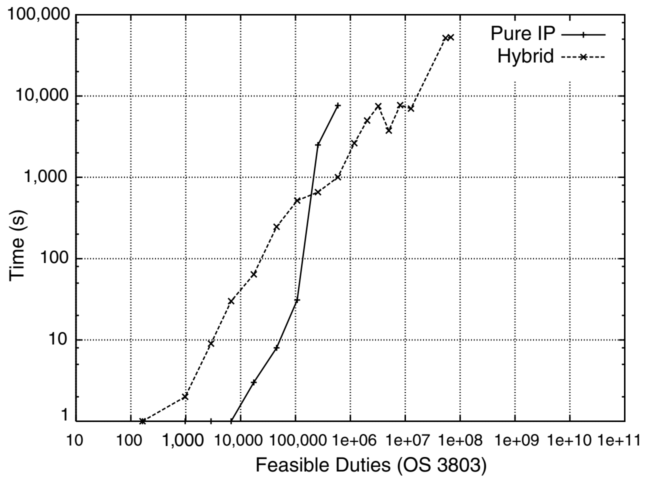

Figure 4 Hybrid Column Generation vs. Integer Programming Over OS 3803

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 that 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 of customer demand. For each case, the crew scheduling problem gives a number of distinct sets of duties as input for the rostering problem.

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 of 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 do we select the 2 crews that are going to be left out? Furthermore, suppose that, after eliminating those crews that cannot work in 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, 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, because some constraints refer to a time window that spans more than one month (see §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 are the following:

• The sets of duties D_{wk}, D_{sa}, D_{su}, and D_{ho} that have to be performed on weekdays, Saturdays, Sundays, and 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 (due to vacations or sickness), excluding's 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 with 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 seven 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 required. In addition, it is desirable to have balanced workloads among all the crews, but the models we present in this article are not concerned with this issue.

2.4. The Format of the Input Datasets

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 §1.

Using the duties built during the crew scheduling phase, we have constructed a set of instances ranging from small to large that are typically encountered by the management personnel in the bus company. The main features of these instances appear in Table 6. 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 indicates 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 is how many of the tt duties are split-shift duties. To begin, we set the following parameters for every crew i: $OFF_i = \varnothing$, $ls_i = 0$, $wr_i = 1$, $sl_i = 0$, 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 IP 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., $|D_{\mathrm{wk}}| = p$, $|D_{\mathrm{sa}}| = q$, $|D_{\mathrm{su}}| = r$, and $|D_{\mathrm{ho}}| = s$). The IP formulation of the rostering problem is based on x_{ijk} binary variables that

Table 6 **Description of the Instances for the Experiments**

			# Duties			
Name	# Crews	# Days	Week	Saturday	Sunday	Holiday
string	c	$d(h)$	$ss_{\mathrm{wk}}/tt_{\mathrm{wk}}$	$ss_{\mathrm{sa}}/tt_{\mathrm{sa}}$	$ss_{\mathrm{su}}/tt_{\mathrm{su}}$	$ss_{\mathrm{ho}}/tt_{\mathrm{ho}}$

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

The main objective is to minimize the number of crews working during the present month. This is equivalent to maximizing the number of crews that are idle during the whole month. Given this choice of variables and objective function, an IP model for this problem can be easily written. Our complete formulation is presented in detail in Yunes et al. (2000b). The linear relaxations and the integer programs were solved with the CPLEX solver. This model appears to suffer from symmetry, as discussed in Barnhart et al. (1998), and finding optimal solutions, even for small instances, turned out to be a difficult task. For example, when looking for optimal solutions for instances with tens of duties in a workday, not even a feasible answer could be found within 30 minutes of computation time. Therefore, we decided to experiment with a pure CLP formulation of the problem.

2.6. A CLP Approach

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

Let n, d, p, q, r, and s be defined as in §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 $i \in \{1, \ldots, n\}$, and $j \in \{1, \ldots, d\}$. The X_{ij}'s are finite domain variables whose domains depend on the value of j. For the complete details see Yunes et al. (2000b).

2.6.1. Computational Results. When compared to the IP model of §2.5, this model performed much better both in terms of solution quality and computation time. As can be seen in Table 7, 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. The column "LB" in Table 7 shows the LP lower bound.

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, we easily found the model solutions. 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. Because we did not have access to the input datasets 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 the severity of the influence of the constraints (2.2) introduced in §2.2. Moreover, we can see from Table 7 that the values of the solutions grow quickly as the number of split-shift duties increases. With this point 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 fewer crews than it did for instance s19. This strengthens the remark made by Caprara et al. (1997) that, ideally, the scheduling and rostering phases should work cyclically, with some feedback between them.

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

2.7. Proving Optimality

In §§2.5 and 2.6, we showed that it is difficult to find provably optimal solutions for this rostering problem. Moreover, it is possible to see from Table 7 that the lower bound provided by the LP relaxation of the problem is always equal to 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, 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, where n is

Table 7 Computational Experiments with the CLP Model

			# Duties						
Name	# Crews	# Days	Week	Saturday	Sunday	Holiday	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

the number of rosters:

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

All numbers a_{ij} in the coefficient matrix A are 0 or 1 and its columns are constructed as shown in Figure 5. Each column is composed of d sequences of numbers, where d is the number of days in the planning horizon. For each $k \in \{1,\ldots,d\}$, the kth 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 rows e, in A, equals $\sum_{k=1}^{d} h_k$.

A column in A must represent a feasible roster for one crew. More precisely, let $t = (u_1, u_2, \ldots, u_d)$ be a

$$\big(\overbrace{0\cdots010\cdots0}^{h_1}\overbrace{0\cdots010\cdots0}^{h_2}\cdots\overbrace{0\cdots0}^{h_i}\cdots\overbrace{0\cdots010\cdots0}^{h_d}\big)^{\mathrm{T}}$$

Figure 5 A Column in the Coefficient Matrix of the Set Partitioning Formulation

feasible roster for a crew, where u_k, $k \in \{1,\ldots,d\}$ is the number of the duty performed on day k. Remember from §2.5 that $u_k \in D_k \cup \{0\}$, where D_k is equal to $\{1,\ldots,p\}$, $\{p+1,\ldots,p+q\}$, $\{p+q+1,\ldots,p+q+r\}$, or $\{p+q+r+1,\ldots,p+q+r+s\}$, 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 ith number will be equal to 1 ($i \in \{1,\ldots,h_k\}$) if and only if u_k is the ith 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 that 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 deemed hybrid because the column generation subproblem is solved by a CLP model. The whole algorithm follows the same basic ideas

Table 8 Computational Experiments with the Hybrid Model

			# Duties						
Name	# Crews	# Days	Week	Saturday	Sunday	Holiday	LB	Opt	Time
s01	10	10 (1)	00/04	00/01	00/01	00/01	4	5	0.66
s02	10	15 (2)	00/04	00/01	00/01	00/01	4	5	2.12
s03	10	20 (2)	00/04	00/01	00/01	00/01	4	5	4.56
s04	10	25 (2)	00/04	00/01	00/01	00/01	4	5	16.72
s13a	15	10 (1)	00/10	00/06	00/05	00/05	10	13	12.73
s18a	30	10 (1)	00/20	00/10	00/10	00/10	20	25	00:04:03
s20a	40	7 (1)	06/26	02/15	02/15	02/15	26	26	21:23:36
s21a	36	7 (1)	00/31	00/20	00/20	00/20	31	31	05:39:50

described in §1.6. The model for the column generator is a variation of the CLP model of §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 cost.

2.7.2. Preliminary Results. The best results for the hybrid model were achieved when initializing the columns of matrix A as the ones obtained by the first solution found by the CLP model of §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 8, where column "LB" gives the LP bound at the root node and 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 §2.5, which was not able to find any optimal solution, even for the smallest instances. Besides, when comparing Tables 7 and 8, 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 is because the IP formulation of §2.7.1 may lead to degenerate problems, as is the case with many partitioning problems with unit cost functions. When trying to solve larger instances, the pricing subroutine keeps on generating columns for a long time, with little or no improvement in the value of the objective function. 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. For example, Figure 6 shows the progress on the value of the objective function in terms of the number of column generation iterations for instance s20a. Notice that it took more than 3,737 iterations to decrease the objective function from the initial value of 34 to something below 33. In this case, the solution of the first LP relaxation was already integer. Out of the 21.4 hours of computation, 20.1 hours (94%) were spent generating columns and 1.3 hours (6%) were spent solving linear programs.

As can be seen in Figure 6, the stalling on the value of the cost function occurs during the first few thousand iterations. Recall that, at each iteration, the CLP solver returned the first 50 columns with negative reduced cost that it could find. As the iterations progressed, the dual variables tended to contain more information, thus helping the CLP solver produce better columns.

Another problem concerns the labeling policy that 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 of finding provably optimal solutions for larger instances of the rostering problem. These modifications are outlined below.

Because 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. We decided then to implement a strategy similar to what was presented in Grötschel et al. (1996) and Uchôa and Poggi de Aragão (1999). The basic idea is to add a small perturbation, $\varepsilon \in [-\delta, \delta]$, to the cost of each column. For

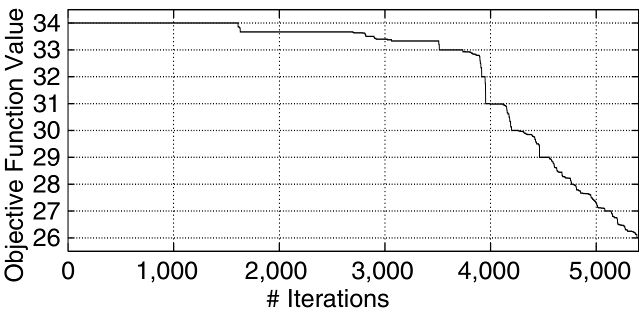

Figure 6 Stalling of the Objective Function Value on Instance s20a

this mechanism to function correctly, the value of δ was chosen according to a simple rule: 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,

$$\delta < \frac{1}{2k+1}. \tag{4}$$

Because 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 (4). 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, because 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 v discrete values and then use a mod-type hash function to map each column to a specific value of perturbation ε. Cormen et al. (1990) suggest that v should be a prime number not too close to a power of 2. We decided to set $v = 1{,}531$.

Set Covering Formulation. With the problem constraints described in §2.2, it is easy to see that any subroster of a feasible roster is itself another feasible roster. Hence, if we change the set partitioning formulation of §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 §1.4.2 that the reduced cost constraint for column c reads

$$\sum_{j \in D_c} u_j > \text{Cost}_c, \tag{5}$$

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 (5). In other words, after choosing one variable to label next, the values in its domain are initially sorted according to the nonincreasing order of their corresponding u_j values. That is, the duties with the largest corresponding dual values are tried first. Because 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.

Looking for Better Dual Prices. We tried different pricing strategies available in CPLEX such as the variants on dual steepest-edge pricing. Also, we tried to use a barrier method to solve the LP relaxation, since these methods have a tendency to converge to strictly complementary and, therefore, nondegenerate solutions with more useful dual prices.

2.7.4. Computational Results with the Improved Algorithm. The inclusion or exclusion of each one of the previous suggested improvements leads to various 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, and with CPLEX's default pricing rule. The results are summarized in Table 9. 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

Table 9 Computational Results with the Best Configuration of the Improved Hybrid Model

			# Duties						
Name	# Crews	# Days	Week	Saturday	Sunday	Holiday	LB	Opt	Time
s01	10	10 (1)	00/04	00/01	00/01	00/01	4	5	0.31
s02	10	15 (2)	00/04	00/01	00/01	00/01	4	5	0.47
s03	10	20 (2)	00/04	00/01	00/01	00/01	4	5	0.62
s04	10	25 (2)	00/04	00/01	00/01	00/01	4	5	0.73
s05	10	30 (2)	00/04	00/01	00/01	00/01	4	5	0.85
s06	10	30 (2)	01/04	00/01	00/01	00/01	4	5	0.89
s07	10	30 (2)	02/04	00/01	00/01	00/01	4	5	0.87
s13a	15	10 (1)	00/10	00/06	00/05	00/05	10	13	7.34
s18a	30	10 (1)	00/20	00/10	00/10	00/10	20	25	20.05
s20a	40	7 (1)	06/26	02/15	02/15	02/15	26	26	12:40:42
s21a	36	7 (1)	00/31	00/20	00/20	00/20	31	31	00:17:19

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 9 reports more than 12 hours of computation for the same instance.

When comparing Tables 8 and 9, 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 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 models. We have given a detailed description of urban transit crew management problems that are part of the daily operation of a medium-size 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. It was difficult to obtain provably optimal solutions for these problems for both IP and CLP approaches when taken in isolation. Our hybrid methodology combines the strengths of both sides, while overcoming some of 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 (Ginsberg 1993) and the use of *nogoods* (Lever et al. 1995) can be applied to traverse the search tree more efficiently, thereby avoiding useless work.

Acknowledgments
The authors would like to thank the Pioneira Bus Company from the city of Belo Horizonte, Brazil, for kindly providing the real-world experimental data. Thanks are also due to the anonymous referees and to the associate editor for many insightful comments that helped improve the quality of this paper. The first author was supported by grants from FAPESP (98/05999-4) and CAPES. The second author was supported by FAPESP (98/05999-4) and FAEP/UNICAMP (0871/02, 0211/01). The third author was supported by FINEP (ProNEx 107/97) and CNPq (300883/94-3).

References

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

Bianco, L., M. Bielli, A. Mingozzi, S. Ricciardelli, M. Spandoni. 1992. A heuristic procedure for the crew rostering problem. *Eur. J. Oper. Res.* **58**(2) 272–283.

Caprara, A., M. Fischetti, P. Toth. 1999. A heuristic method for the set covering problem. *Oper. Res.* **47**(5) 730–743.

Caprara, A., M. Fischetti, P. Toth, D. Vigo. 1998. Modeling and solving the crew rostering problem. *Oper. Res.* **46**(6) 820–830.

Caprara, A., M. Fischetti, P. Toth, D. Vigo, P. L. Guida. 1997. Algorithms for railway crew management. *Math. Programming* **79** 125–141.

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

Carraresi, P., G. Gallo. 1984. A multi-level bottleneck assignment approach to the bus drivers' rostering problem. *Eur. J. Oper. Res.* **16**(2) 163–173.

Caseau, Y., P. Koppstein. 1992. A rule-based approach to a time-constrained traveling salesman problem. *2nd Internat. Sympos. Artificial Intelligence Math.*

Chabrier, A. 1999. A cooperative CP and LP optimizer approach for the pairing generation problem. *Internat. Workshop Integration AI & OR Tech. Constraint Programming Combinatorial Optim. Problems* (*CP-AI-OR'99*), Ferrara, Italy.

Cheng, B. M. W., K. M. F. Choi, J. H. M. Lee, J. C. K. Wu. 1999. Increasing constraint propagation by redundant modeling: An experience report. *Constraints* **4**(2) 167–182.

Cormen, T. H., C. E. Leiserson, R. L. Rivest. 1990. *Introduction to Algorithms*. MIT Press, Cambridge, MA.

Crainic, T. G., G. Laporte, eds. 1998. *Fleet Management and Logistics*. Kluwer Academic Publishers, Boston, MA.

Darby-Dowman, K., J. Little. 1998. Properties of some combinatorial optimization problems and their effect on the performance of integer programming and constraint logic programming. *INFORMS J. Comput.* **10**(3) 276–286.

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

Dror, M., ed. 2000. *Arc Routing: Theory, Solutions and Applications*. Kluwer Academic Publishers, Boston, MA.

Fahle, T., M. Sellmann. 2000. Constraint programming based column generation with knapsack subproblems. 2*nd Internat. Workshop Integration AI & OR Tech. Constraint Programming Combinatorial Optim. Problems* (*CP-AI-OR'*00), Paderborn, Germany.

Gervet, C. 1998. Large combinatorial optimization problems: A methodology for hybrid models and solutions. *Journées Francophones de Programmation en Logique et par Contraintes*, Nantes, France.

Ginsberg, M. L. 1993. Dynamic backtracking. *J. Artificial Intelligence Res.* **1** 25–46.

Grötschel, M., A. Martin, R. Weismantel. 1996. Packing Steiner trees: A cutting plane algorithm and computational results. *Math. Programming* **72** 125–145.

Guerinik, N., M. V. Caneghem. 1995. Solving crew scheduling problems by constraint programming. *Lecture Notes Comput. Sci.* **976** 481–498. 1*st Internat. Conf. Principles Practice Constraint Programming*, *CP'*95, Cassis, France.

IC-Parc. 2001. Imperial College, London, The ECL^{i}PS^e Constraint Logic Programming System, Ver. 5.1.3. http://www.icparc.ic.ac.uk/eclipse.

ILOG S. A. 1999. *ILOG CPLEX 6.5 Reference Manual*.

Jachnik, J. K. 1981. Attendance and rostering system. *Computer Scheduling of Public Transport*. A. Wren, ed. North-Holland Publishing Company, Amsterdam, The Netherlands, 337–343.

Jourdan, J. 1995. Concurrent constraint multiple models in CLP and CC languages: Toward a programming methodology by modeling. Doctoral dissertation, Université Denis Diderot, Paris VII, France.

Junker, U., S. Karisch, N. Kohl, B. Vaaben. 1999a. Constraint programming based column generation for crew assignment. *Internat. Workshop Integration AI OR Tech. Constraint Programming for Combinatorial Optimization Problems* (*CP-AI-OR'*99), Ferrara, Italy.

Junker, U., S. Karisch, N. Kohl, B. Vaaben, T. Fahle, M. Sellmann. 1999b. A framework for constraint programming based column generation. *Lecture Notes Comput. Sci.* **1713** 261–274. 5*th Internat. Conf. Principles Practice Constraint Programming* (*CP'*99), Alexandria, VA.

Leconte, M., P. Couronne, D. Vergamini. 1996. Column generation using cooperating constraint-based solvers. *Workshop on Constraints and Constraint Programming, Asian Computing Sci. Conf. (Asian'96)*, Singapore.

Lever, J., M. Wallace, B. Richards. 1995. Constraint logic programming for scheduling and planning. *British Telecom Tech. J.* **13** 73–81.

OREAS GmbH. 1999. ABACUS: A Branch-And-CUt System, Ver. 2.3. *User's Guide and Reference Manual*.

Reeves, C. R., ed. 1993. *Modern Heuristic Techniques for Combinatorial Problems*. Wiley, New York.

Sellmann, M., K. Zervoudakis, P. Stamatopoulos, T. Fahle. 2000. Integrating direct CP search and CP-based column generation for the airline crew assignment problem. 2*nd Internat. Workshop Integration AI & OR Techniques Constraint Programming Combinatorial Optim. Problems* (*CP-AI-OR'*00), Paderborn, Germany.

Uchôa, E., M. Poggi de Aragão. 1999. Vertex-disjoint packing of two Steiner trees: Polyhedra and branch-and-cut. *Lecture Notes Comput. Sci.* **1610** 439–452. 7*th Internat. Conf. Integer Programming Combinatorial Optim.* (*IPCO'*99), Graz, Austria.

Yunes, T. H., A. V. Moura, C. C. de Souza. 1999. Exact solutions for real world crew scheduling problems. *INFORMS Fall* 1999 *Meeting*, Philadelphia, PA.

Yunes, T. H., A. V. Moura, C. C. de Souza. 2000a. A hybrid approach for solving large scale crew scheduling problems. *Lecture Notes Comput. Sci.* **1753** 293–307. 2*nd Internat. Workshop Practical Aspects Declarative Languages* (*PADL'*00), Boston, MA.

Yunes, T. H., A. V. Moura, C. C. de Souza. 2000b. Hybrid column generation approaches for solving real world crew management problems. Technical Report 00-18, Institute of Computing, University of Campinas (UNICAMP). Campinas, SP, Brazil. http://goa.pos.ic.unicamp.br/otimo/pubtexts/rt-00-18.ps.gz.