Solving Very Large Crew Scheduling Problems to Optimality

Tallys H. Yunes*
Institute of Computing,
University of Campinas
tallys@acm.org

Arnaldo V. Moura
Institute of Computing,
University of Campinas
P.O. Box 6176, ZIP 13083-970
Campinas, SP, Brazil
arnaldo@dcc.unicamp.br

Cid C. de Souza†
Institute of Computing,
University of Campinas
cid@dcc.unicamp.br

Keywords

Crew Scheduling, Constraint Programming, Mathematical Programming, Hybrid Algorithms, Column Generation

ABSTRACT

In this article, we present a hybrid methodology for the exact solution of large scale real world crew scheduling problems. Our approach integrates mathematical programming and constraint satisfaction techniques, taking advantage of their particular abilities in modeling and solving specific parts of the problem. An Integer Programming framework was responsible for guiding the overall search process and for obtaining lower bounds on the value of the optimal solution. Complex constraints were easily expressed, in a declarative way, using a Constraint Logic Programming language. Moreover, with an effective constraint-based model, the huge space of feasible solutions could be implicitly considered in a fairly efficient way. Our code was tested on real problem instances arising from the daily operation of an ordinary urban transit company that serves a major metropolitan area with an excess of two million inhabitants. Using a typical desktop PC, we were able find, in an acceptable running time, an optimal solution to instances with more than 1.5 billion entries.

1. INTRODUCTION

Crew scheduling problems have their great practical importance based on the fact that, in most companies, employee related expenses may rise to a very significant portion of the total expenditures. Therefore, these notoriously difficult combinatorial optimization problems deserve a great deal of attention. In this article, we present a hybrid strategy that is capable of efficiently obtaining provably optimal solutions for some large instances of specific crew scheduling problems. These instances stem from the operational environment of a mass transit company that serves the metropolitan area of the city of Belo Horizonte, in Brazil.

Previous attempts to solve similar crew scheduling problems to optimality, either with Integer Programming (IP) or with Constraint Programming (CP) techniques alone, faced difficulties when handling large scale instances due to a series of factors [2; 6; 8].

In order to combine the capabilities of both the IP and CP techniques, we developed a hybrid approach to solve 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 the dual variables in the solution of the linear relaxation, we can enter a column generation phase that computes new feasible duties. This phase is modeled as a constraint satisfaction problem which is then submitted to a constraint solver. The solver returns new feasible duties to be inserted in the IP problem formulation, and the initial phase can be taken again, restarting the cycle. This approach secures the strengths of both the pure IP and the pure CP formulations: 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 with an excess of 150 trips and 12 million feasible duties. The resulting code was compiled under the Linux operating system, kernel 2.0, and ran on a 350 MHz desktop PC with 320 MB of main memory.

This article is organized as follows. Section 2 describes the crew scheduling problem. In Section 3, we discuss some difficulties faced by the pure IP and CP approaches. In Section 4, we present the hybrid approach together with some implementation details, and also report on computational results based on real data. Finally, in Section 5, we conclude and discuss further issues.

2. THE CREW SCHEDULING PROBLEM

In a typical crew scheduling problem, a set of trips has to

*Supported by FAPESP grant 98/05999-4, and CAPES.
†Supported by FINEP (ProNEx 107/97), and CNPq (300883/94-3).

SAC '00 Villa Olmo, Como, Italy

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.

2.1 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 *depot* is a location where crews may change buses and rest. The act of driving a bus from one-depot to another depot, passing by no intermediate depots, is named a *trip*. Associated with a trip we have its *start time*, its *duration*, its *initial depot*, and its *final depot*. 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. Any time interval between two consecutive trips in a duty is called an *idle interval*. Whenever this idle interval exceeds *Idle_Limit* minutes, it is called a *long rest*. During a long rest, crews can leave the premises and return later to resume their shift. A duty that contains a long rest is called a *split-shift duty*. The *rest time* of a duty is the sum of its idle intervals, 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 time, in minutes, that a crew member works in excess of *Workday* minutes is called *overtime* and is given by $\max\{0, \textit{total working time} - \textit{Workday}\}$. The *Workday* is a given parameter, specified by union regulations, that bounds the maximum time that an employee can work without incurring in overtime. An upper bound on *overtime* is established by the parameter *Max_Overtime*. Finally, the *maximum working time* is given by $\textit{Workday} + \textit{Max\_Overtime}$.

2.2 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 four columns with information about this trip: *start time*, measured in minutes after midnight, *duration*, measured in minutes, *initial depot* and *final depot*. We have used data that reflect the operational environment of two bus lines, Line 2222 and Line 3803, that serve a major metropolitan area. Line 2222 has 125 trips and one depot and Line 3803 has 246 trips and two depots. The input data tables for these lines are called OS 2222 and OS 3803, respectively. 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. Table 1(a) shows the first 10 rows of OS 3803. 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 graphic was constructed as follows. For each (x, y) entry, we consider a time window $T = [x, x + \textit{Workday}]$. The ordinate y indicates how many trips there are with start time s and duration d such that $s \in T$ or $s + d \in T$. The particular shapes of these two curves represent a typical daily workload in the operation of an urban bus company in Brazil.

2.3 Constraints

For a duty to be classified as feasible, it has to satisfy many constraints imposed by labor contracts and union regulations, among others. The most important constraints are, for every duty:

i. For each pair of consecutive trips, i and j:
   (i) $(\textit{start time})_i + (\textit{duration})_i \leq (\textit{start time})_j$
   (ii) $(\textit{final depot})_i = (\textit{initial depot})_j$
ii. $\textit{total working time} \leq \textit{maximum working time}$;
iii. $\textit{rest time} + \max\{0, \textit{Workday} - \textit{total working time}\} \geq \textit{Min\_Rest}$; and
iv. At most one long rest interval is allowed;

Due to union regulations and operational constraints, the following values were used in our experiments: $\textit{Idle\_Limit} = 120$, $\textit{Workday} = 440$, $\textit{Min\_Rest} = 30$ and $\textit{Max\_Overtime} = 0$, measured in minutes. A duty which satisfies all problem constraints is called a *feasible duty*. Any set of feasible duties constitutes a *schedule* and for a schedule to be *acceptable* it must partition the set of trips. The cost of a schedule is the sum of the costs of all its duties. As we are interested in minimizing the number of crews needed to operate the bus line, all duties are treated equally and their costs are set to one. With this assumption, minimizing the cost of a schedule reduces to minimizing the number duties (crews) in the solution. Finally, a *minimal schedule* is any acceptable schedule with minimum cost.

3. PURE APPROACHES

The crew scheduling problem, as described here, presents a classical difficulty: finding the optimal schedule reduces to choosing from an extremely large set F of feasible duties, a minimal subset that partitions the set of trips to be serviced. The very large size of F poses serious problems, since a provably optimal solution can only be found if all elements of F are considered, either implicitly or explicitly.
When adopting IP approaches to solve this problem, one usually ends up with a set partitioning formulation where the elements of F constitute the columns of the coefficient matrix. The pure IP formulation of the problem is:

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

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

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

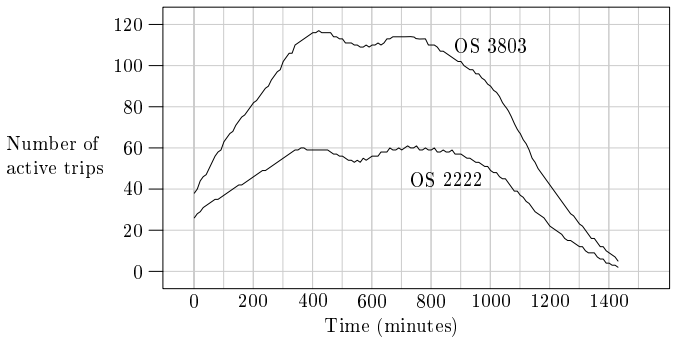
where m is the total number of trips and n is the size of F. The x_j's are 0-1 decision variables that indicate which duties belong to the solution. The coefficient a_{ij} equals 1 if duty j contains trip i, otherwise, a_{ij} is 0.
Due to the size of F, an implicit way to treat the set of feasible duties is needed. Column generation [1] is a technique that is widely used to handle linear programs which have a very large number of columns in the coefficient matrix. 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. This problem is called the *master* problem, and deals only with a subset of the original columns. That smaller problem solved, we

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)



start phase two, looking for columns with a 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 costs, and start over on phase one. From the 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 serviced by 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 are restricted to integer values, this algorithm must be embedded in a branch-and-bound strategy. The resulting algorithm is usually referred to as *branch-and-price*. The slave subproblem can be modeled as a constrained shortest path problem over a directed acyclic graph and it can be solved by a dynamic programming algorithm [3]. Nevertheless, the pseudo-polynomial algorithms that arise from this strategy turn out to be very time consuming. This is mainly because of the looseness of our problem constraints [8].

Difficulties also arise when trying to solve crew scheduling problems using pure constraint satisfaction techniques [2; 6; 8]. The huge search space can only be dealt with efficiently 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 naturem to optimality, none of the these requirements can be easily met, rendering it intrinsically difficult for pure CP techniques to produce satisfactory results in these cases.

4. A HYBRID APPROACH

Recent research [4] has shown that, in some cases, neither the pure IP nor the pure declarative CP approaches are capable of solving certain kinds of combinatorial problems satisfactorily. But a hybrid strategy may outperform these two methods. When contemplating a hybrid strategy, it is necessary to to decide which part of the problem will be handled by a constraint solver, and which part will be dealt with in a classical way. Given the huge number of columns at hand, the use of a column generation approach seemed to be almost mandatory.

A declarative constraint language is particularly suited to express the feasibility constraints imposed by the original problem in a clear and concise way. Furthermore, a constraint model of the original problem can also be easily turned into an efficient column generator by adding one extra constraint to the model: the generated duties must have a negative reduced cost so as to improve the objective function value [1]. Moreover, when looking for new feasible duties (columns) to enter the basis in the current set partitioning formulation, it is not necessary to find the one with *the* most negative reduced cost. This removes the minimization constraint from the formulation, rendering it much more efficient. Our hybrid strategy implemented a column generation approach where new columns were generated on demand by a declarative constraint program.

We decided to use the ABACUS[1] branch-and-price framework in order to save programming time. The Linear Programming relaxations of the set partitioning formulation of the problem are solved inside ABACUS with the help of a CPLEX[2] 3.0 LP solver. When the ABACUS process has solved the current master problem to optimality, it sends the values of the dual variables to the constraint solver, together with the number of columns with negative reduced costs it would like to get. If there remained some such columns, a subset of them is captured by the CP solver and 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 the current 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 at a different node of the implicit enumeration tree. By experimenting with the data sets at hand, we determined that the number of columns with negative reduced cost to request at each call of the CP solver was best set to 53.

4.1 The Column Generator

Modeling with finite domain constraints is rapidly gaining acceptance as a promising declarative programming environment to solve large combinatorial problems. All models described in this section were formulated using the ECL^{i}PS^e [3] syntax, version 4.0. Due to its large size, the ECL^{i}PS^e formulation for each run was prodiced by a program generator that we developed in C.

When creating the constraint-based column generator, our

[1] `http://www.informatik.uni-koeln.de/ls_juenger/`

[2] CPLEX is a registered trademark of ILOG, Inc.

[3] `http://www.icparc.ic.ac.uk/eclipse`

aim was to facilitate the direct representation of algebraic constraints. The model is based on a vector X of integers. The number of elements in X is an upper bound on the size of any feasible duty ($UBdutyLen$). 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. Each X_i element, called a *cell*, represents a single trip and is a finite domain variable with domain $[1..NT]$, where $NT = N + UBdutyLen - 1$ and N is the number of real trips. Trips numbered $N+1$ to NT are *dummy trips*. 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 trips, i.e., they start at each following minute. Their departure and arrival depots are equal to 0. These choices prevent incompatibilities arising from time intersection and mismatching of depots among the dummy trips. Besides, we avoid the occurrence of dummy trips between real trips in a feasible duty.

Note that, the way $UBdutyLen$ is calculated assures that at least one dummy trip appears in X. Moreover, their start times guarantee that the dummy trips occupy consecutive cells at the end of X. This is on purpose, to facilitate the representation of some constraints.

Five additional vectors were used: *Start*, *End*, *Dur*, *DepDepot* and *ArrDepot*. The i-th cell of these vectors represents, respectively, the start time, the end time, the duration, and the departure and arrival depots of the trip assigned to X_i. Next, we state constraints of type $\texttt{element}(X_i, S, Start_i)$, where S is a list containing the start times of all NT trips. The semantics of this constraint assures that the value of $Start_i$ is the k-th element of list S where k is the value in X_i. This maintains the desired relationship between vectors X and *Start*. Whenever X_i is updated, e.g. due to constraint propagation, $Start_i$ is also modified, and vice-versa. Similar constraints are stated between X and each one of the four other vectors. Now, we can use these new vectors to easily state additional constraints, like:

$$End_i \leq Start_{i+1} \quad (4)$$

$$ArrDepot_i + DepDepot_{i+1} \neq 3 \quad (5)$$

$$Idle_i = BD_i \times \big(Start_{i+1} - End_i\big) \quad (6)$$

for all $i \in \{1, \ldots, UBdutyLen - 1\}$. Equation (4) guarantees that trips overlapping in time are not in the same duty. Since the maximum number of depots is two, an incompatibility between consecutive trips occurs only when the ending depot is 1 and the starting depot is 2, or vice-versa. Equation (5) forbids that situation. Additionally, the consecutiveness of two dummy trips is permitted (for the sum of their depots equals 0) and the appearance of the first dummy trip after the last real trip in a duty is not precluded by this constraint, because the sum of the depots in this case can only assume the values 1 or 2. With a one-depot instance, these constraints are not necessary and they are omitted, together with the *ArrDepot* and *DepDepot* vectors. Some other constraints are expressed using the $Idle_i$ variables of Equation (6). The binary variables BD_i, in (6), are such that $BD_i = 1$ if and only if X_{i+1} contains a real trip.

The constraint on total working time, for each generated duty, is given by:

$$TWT = \sum_{i=1}^{UBdutyLen-1} Dur_i + \sum_{i=1}^{UBdutyLen-1} BI_i \times Idle_i \quad (7)$$

$$TWT \leq \textit{maximum working time} \quad (8)$$

where BI_i is a binary variable such that $BI_i = 1$ if and only if $Idle_i \leq Idle\_Limit$.

The constraint on total rest time is:

$$\sum_{i=1}^{UBdutyLen-1} Idle_i + \max\{0, Workday - TWT\} \geq Min\_Rest\,. \quad (9)$$

To respect the constraint on split-shift duties, we impose that at most one of the $Idle_i$ variables can assume a value greater than $Idle\_Limit$. This is done with an `atmost` constraint in the following manner: $\texttt{atmost}(1, L, 0)$. If list L contains all the BI_i variables of Equation (7), this means that at most one of them can assume the value zero.

Changing dummy trips in a feasible duty gives another duty that is equivalent to the original one. New constraints were imposed over the X cells in order to force dummy trips to have only one possible placement in X, given that the real trips had already been positioned. This is achieved with the following constraints, for all $i \in \{1, \ldots, UBdutyLen - 1\}$, where N is the number of real trips:

$$X_i \leq N \Rightarrow (X_{i+1} > N \Rightarrow X_{i+1} = N + 1) \quad (10)$$

$$X_i > N \Rightarrow X_{i+1} = X_i + 1\,. \quad (11)$$

There is one final constraint, which is responsible for assuring that generated duties have an associated negative reduced cost. Using the formula to calculate the reduced cost of a column (feasible duty) given in Section 3, this constraint reads:

$$\sum_{i=1}^{UBdutyLen} C_i > 1\,. \quad (12)$$

For each i, C_i is determined by $\texttt{element}(X_i, V, C_i)$, where V 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.

Various labeling strategies have been tried. The strategy of choosing the next variable to label as the one with the smallest domain (*first-fail* principle) proved to be the most effective one, after a number of experimental trials. Having chosen 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 produced the best results. It would be possible to improve the overall performance of this search strategy by experimenting with more sophisticated techniques. Dynamic backtracking [5] and *nogood* assertions [7] could be used in this model so as to promote an earlier pruning of unpromising branches of the search tree.

A crucial advantage of this hybrid approach over a number of previous attempts is that it considers *all* feasible duties. Therefore, it is not necessary to use specific rules to select, at the beginning, a subset of "good" feasible duties. This kind of preprocessing could prevent the optimal solution from being encountered. Instead, with the column

Table 2: OS 2222 data set (1 depot)

#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

Table 3: OS 3803 data set (2 depots)

#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

generation method, our algorithm implicitly looks at the set of all feasible duties.

4.2 Computational Results

In this section, execution times inferior to one minute are reported as *ss.cc*, where *ss* denotes seconds and *cc* denotes hundredths of seconds. When execution times exceed 60 seconds, we use the alternative notation *hh*:*mm*:*ss*, where *hh*, *mm* and *ss* represent, respectively, hours, minutes and seconds.

When compared to the pure approaches, the hybrid approach was able to construct an optimal solution to substantially large instances of the problem, in a reasonable time. Computational results for OS 2222 and OS 3803 appear on Tables 2 and 3, respectively. Column headings have the following meanings: #Trips is the number of trips; #FD stands for the number of feasible duties; Opt is the value of the optimal integer solution; DBR is the dual bound at the root node of the branch-and-bound enumeration tree; #CA is the number of columns added throughout each execution; #LP is the number of linear programming relaxations solved; and #Nodes is the number of tree nodes visited. The execution times are divided in three columns: PrT is the time spent generating columns; LPT is the time spent solving linear programming relaxations and TT is the total execution time. Note that, in every instance tested, 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. 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 added columns.

It is interesting to note, in the last three columns of each table, that the time taken to solve all linear relaxations of the problem was a small fraction of the total running time for both data sets.

From Table 2, we see that the hybrid approach was capable of constructing a provably optimal solution for the complete smaller data set using 21 minutes of running time on a 350 MHz desktop PC. That problem involves in excess of one million feasible duties.

The structural difference between both data sets can be appreciated by observing the entries on the line associated with 100 trips, in Table 3. The number of feasible duties on this line corresponds, approximately, to the same number of 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 when running over a test case that contained the first 100 trips of the larger data set, as it did when taking the 125 trips of the smaller data set. The existence of two depots in OS 3803, rather than a single one in OS 2222, seems to be at the origin of the increased problem complexity of this instance.

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 to reach completion.

A problem instance with as many as $150 \times (12.5 \times 10^6)$ entries would require over 1.8 GB of main memory, if needed to be loaded into main memory. By efficiently dealing with only a small subset of the feasible duties, our algorithm managed to surpass the resource consumption bottlenecks faced by the pure approaches and could solve instances that were very large. This observation supports our view that a declarative constraint formulation of column generation, embedded inside a branch-and-bound framework, was the right technique to solve these very large crew scheduling problems.

5. CONCLUSIONS AND FUTURE WORK

We have integrated pure Constraint Programming and pure Integer Programming techniques in a hybrid column generation algorithm that is able to solve very large instances of some real world crew scheduling problems to optimality. These problems appear intractable for both approaches when taken in isolation. Our methodology combines the strengths of both methods, getting over their main weaknesses.

Complex operational constraints could easily be expressed using declarative statements in a constraint programming language. The resulting model proved to be very efficient when looking for feasible duties with a negative reduced cost. A linear programming relaxation of the original problem formulation produced good lower bounds. Combining these two behaviors, it was possible to develop a very successful branch-and-price scheme. Using this hybrid method, a desktop PC was able to find optimal solutions for large one-depot instances in a reasonable time. The two-depot case, however, having a different structure and a larger search space, presented more difficulties. When we restricted the problem to the first 150 trips of the day, which already correspond to 12 million feasible duties, the hybrid algorithm produced an optimal solution. Nevertheless, even increasing the computation time limit to 24 hours, the hybrid algorithm was not able to compute an optimal schedule to the complete set of trips, which contained an excess of 122 million feasible duties. Further investigation is needed in order to devise more effective strategies to deal with such large two-depot instances.

6. 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. http://tli.isye.gatech.edu/research/papers/papers.htm.

[2] 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):276–286, 1998.

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

[4] C. Gervet. Large Combinatorial Optimization Problems: a Methodology for Hybrid Models and Solutions. In *Journées Francophones de Programmation en Logique et par Contraintes*, 1998. http://www.icparc.ic.ac.uk/papers_byauthor.html.

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

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

[7] J. Lever, M. Wallace, and B. Richards. Constraint logic programming for scheduling and planning. *British Telecom Technical Journal*, (13):73–81, 1995. http://www.icparc.ic.ac.uk/papers_byauthor.html.

[8] T. H. Yunes, A. V. Moura, and C. C. de Souza. Solving large scale crew scheduling problems with constraint programming and integer programming. Technical Report IC-99-19, Institute of Computing, University of Campinas, Brazil, 1999. http://goa.pos.dcc.unicamp.br/otimo/published.html.

APPENDIX

A. ABOUT THE AUTHORS

Tallys H. Yunes got a Computer Engineering degree at the University of Campinas (UNICAMP), Brazil, in 1997. He is currently a M.Sc. student at the Institute of Computing of UNICAMP, Brazil. His main research interests include Combinatorial Optimization, Integer Programming and Constraint Programming.

Arnaldo V. Moura got his Ph.D. degree in Computer Science at the University of California at Berkeley, USA, in 1980. He is currently a full professor at the Institute of Computing of UNICAMP, Brazil. His main research interests include Automata Theory, Formal Languages, Computability, Verification of Hybrid Systems, Formal Methods in Software Engineering and Combinatorial Optimization.

Cid C. de Souza got his Ph.D. degree in Computer Science at the Catholic University of Louvain (CORE), Belgium, in 1993. He is currently a full professor at the Institute of Computing of UNICAMP, Brazil. His main research interests include Combinatorial Optimization, Linear and Integer Programming and Design and Analysis of Algorithms.