Mathematical Programming 81 (1998) 215–228

A Lagrangian-based heuristic for large-scale set covering problems

Sebastián Ceria [a,*,1], Paolo Nobili [b], Antonio Sassano [c,2]

[a] *Graduate School of Business, Columbia University, New York, NY 10027, USA*
[b] *Istituto di Analisi dei Sistemi ed Informatica, CNR, Italy*
[c] *Dipartimento di Informatica e Sistemistica, Università di Roma, La Sapienza, Italy*

Received 30 October 1994; revised manuscript received 15 December 1995

Abstract

We present a new Lagrangian-based heuristic for solving large-scale set-covering problems arising from crew-scheduling at the Italian Railways (Ferrovie dello Stato). Our heuristic obtained impressive results when compared to state-of-the-art codes on a test-bed provided by the company, which includes instances with sizes ranging from 50,000 variables and 500 constraints to 1,000,000 variables and 5000 constraints. © 1998 The Mathematical Programming Society, Inc. Published by Elsevier Science B.V.

Keywords: Set covering; Crew scheduling; Primal-dual lagrangian; Subgradient algorithms; 0-1 programming; Approximate solutions; Railways

1. Introduction

The mathematical programming literature on the crew-scheduling problem is vast in the context of the airline industry (see for example [1–6]), but contains very few examples of problems from the railways. In this paper we present a Lagrangian-based heuristic to solve large-scale set-covering problems arising from crew-scheduling at the Italian railways. The main objective of the crew-scheduling problem is to assign crews so that all train trips are covered at minimum total cost, while respecting working rules. As in the airline case, the train trips are divided into segments or *legs*, and we have the constraint that at least one crew must cover each leg. Train legs are combined into longer trips called *duties* or *pairings* that start and end at a "home-base" and last typically for one or two days. Not all combinations of train legs provide feasible du-

* Corresponding author.

[1] This research was performed while the author was affiliated with IASI, CNR and Dipartimento di Informatica e Sistemistica, Università di Roma, La Sapienza, Italy.

[2] This research was partially supported by National Research Program "Metodi di Ottimizzazione per le Decisioni", MURST, Roma, Italy.

S0025-5610/98/$19.00 © 1998 The Mathematical Programming Society, Inc.
Published by Elsevier Science B.V.
PII S0025-5610(97)00081-6

ties. Indeed, the set of legs must satisfy working rules, which are often written in labor conventions and are the result of lengthy discussions between labor and management.

There are three important differences between scheduling crews in the railway and the airline industries that should be of concern to the reader. First, and probably the most important, is the number of legs that needs to be covered. Indeed, for most algorithms available, this is the most important parameter for the computational complexity. Second, the cost structure is very different. In the railway industry, the cost of different duties is quite similar, in fact, the data we worked on contain duties that only cost one or two units, which correspond to the number of days that the whole trip lasts. This cost structure makes the problem almost combinatorial and more difficult to solve by existing algorithms. On the other hand, in the airline industry, the cost structure of the duties is much less symmetrical, which is an advantage for most algorithms. Last is the fact that most airlines require that each leg be covered exactly once, that is by only one crew (the working crew), while for the railways the number of crews on a train is irrelevant, as long as there is at least one.

Crew-scheduling is only a part of the whole process of scheduling crews to trains. In practice, the crew-scheduling problem is solved as a first step for deciding the final schedule for each crew. The second step is the crew-pairing or crew-rostering problem, where the daily assignments for the crews are combined into weekly or monthly schedules, including working days and rest periods.

Most of the algorithms for crew-scheduling work as follows. First, a large set of feasible duties (we call a duty feasible if it satisfies the working rules) is generated. The generation of duties is usually performed by enumerative techniques, that take into account the complicated working rules and labor restrictions. Then a subset of these duties is selected so that every leg is covered at minimum total cost. This second problem is solved by using a set-covering model, whose columns are defined by the generated pairings. In this paper we mainly deal with the solution of this set-covering problem, although the efficiency of our approach is somewhat dependent on how the first part of the process is carried out.

The data provided to us by the company correspond to real-world instances and consist of large sets of feasible duties generated by Ferrovie dello Stato. The size of the associated set-covering problems range from 50,000 variables and 500 constraints to 1,000,000 variables and 5000 constraints. The average number of non-zero entries per column is about 10 and all costs are either 1 or 2. We believe that the optimal solution of the larger instances is beyond the current state-of-the-art in mathematical programming, and hence these problems have to be attacked with heuristics. The large number of constraints in the larger instances makes linear programming based heuristics computationally ineffective. Indeed, solving the linear programming relaxation of the larger problems would imply an enormous computational effort, in both time and computer memory. Instead, our approach is based on Lagrangian relaxation, a technique that has already proved itself quite effective in dealing with this class of problems.

Several heuristics have been proposed in the literature for the solution of the set-covering problem. Many of these follow a general scheme of a *greedy* character,

which amounts to iteratively choosing a variable which maximizes some merit function, adding such a variable to the current partial solution (setting that variable to 1), and removing the rows which are covered by the column of the matrix corresponding to the variable. The algorithm stops when all the rows have been removed. At this point, the current solution is feasible for SC and can be improved by removing redundant columns. Chvatal [7] proved a worst-case bound for the greedy procedure when the merit function is given by the ratio between the cost and the number of rows covered by a given column in the current subproblem. Balas and Ho [8] and, more recently, Feo and Resende [9] have proposed a number of different merit functions for the greedy procedure.

A more effective family of heuristics is based on the computation of the Lagrangian relaxation of SC. The use of the Lagrangian relaxation for solving the set-covering problem was originally proposed by Balas and Ho [8], who devised an exact branch-and-bound algorithm. They also proposed to exploit the reduced cost information obtained by the subgradient algorithm to produce a feasible solution of SC. Subsequently, Fisher and Kedia [10] proposed an efficient continuous dual heuristic in the more general case of mixed set-covering/set-partitioning constraints. Finally, Beasley [11] proposed to perform reduced cost fixing and computation of a feasible solution at each iteration of the subgradient algorithm. In this way, better results were obtained for problems with up to 1000 rows and 10,000 columns. Recently, Balas and Carrera [12] proposed an integrated upper and lower bounding procedure which combines the standard subgradient method with primal and dual heuristics that interact to change the Lagrange multipliers. They use such a method both as a stand-alone heuristic and as the centrepiece of a branch-and-bound procedure. Their computational results improve upon the previous procedures.

The algorithm we propose in this paper combines these two successful approaches. Namely, it uses the information gathered during the computation of a Lagrangian relaxation to construct a suitable merit function which is later used to select variables in a greedy heuristic.

In the remainder of this section we introduce the basic notation, problem formulation and definitions. In Section 2 we give our main algorithm for the computation of feasible solutions of our problem. We also discuss issues related to the efficient computational implementation for large-scale instances and describe the basic algorithms for computing lower bounds for the associated set-covering problems. Finally, in Section 3 we present computational results for a set of problems given to us by the Italian railways, and on some problems in a publicly available data set.

We first introduce some notation. Let $I = \{S_1, \ldots, S_n\}$ be the set of feasible pairings, and $J = \{1, \ldots, m\}$ the set of train legs that need to be covered by the pairings in I. We will assume that for each $j \in J$, there is at least one pairing $S_i \in I$, covering j. Let $A = [a_{ij}]$ be the $m \times n$ matrix whose ith column is the characteristic vector of the pairing S_i. In other words, $a_{ij} = 1$, if $j \in S_i$, and $a_{ij} = 0$, otherwise. Furthermore, let c_i be the cost of selecting pairing S_i, $i = 1, \ldots, n$. We will assume, without loss of generality, that all c_i's are positive, if not, the constraints covered by such a pairing can

be eliminated from the problem and the pairing included in the solution by setting the corresponding $x_i = 1$. The crew-scheduling problem can then be formulated as the following set-covering problem.

$$\begin{aligned} \min \quad & cx \\ \text{s.t.} \quad & Ax \geqslant \mathbf{1} \\ & x_i \in \{0,1\}, \quad i = 1,\ldots,n, \end{aligned}$$

where $x_i = 1$, if pairing i is chosen, and $x_i = 0$, otherwise and $\mathbf{1}$ is an m-vector of ones.

The linear programming relaxation of SC, that we denote by LSC, is obtained by relaxing the integrality restrictions on x_i, $i = 1,\ldots,n$, and replacing them with simple lower bounding constraints, i.e. $x_i \geqslant 0$, $i = 1,\ldots,n$. Notice that the upper bounding constraints are not needed in the LP relaxation, they are implied by the fact that the objective function is non-negative.

The dual of LSC, denoted by DLSC will also be important for us:

$$\begin{aligned} \max \quad & \mathbf{1}y \\ \text{s.t.} \quad & A^{\mathrm{T}}y \leqslant c, \\ & y \geqslant 0. \end{aligned}$$

2. A new heuristic for the set-covering problem

In order to describe our approach, we first recall some basic notions of the Lagrangian method. We consider the following Lagrangian relaxation of SC:

$$\max \; L(\lambda),$$

where $\lambda \in \mathbb{R}^m_+$, and

$$L(\lambda) = \min \; (c - A^{\mathrm{T}}\lambda)x + \mathbf{1}\lambda,$$

with the minimum taken over $x_i \in \{0,1\}$, $i = 1,\ldots,n$.

Given λ, the value of $L(\lambda)$ can be easily computed by studying the signs of the components of $(c - A^{\mathrm{T}}\lambda)$. In fact, it is easily observed that any vector $\bar{x}$ which minimizes $(c - A^{\mathrm{T}}\lambda)x$ is obtained by setting $\bar{x}_j = 1$, if $(c_j - \sum_{i=1}^m a_{ij}\lambda_i) < 0$, $\bar{x}_j = 0$, if $(c_j - \sum_{i=1}^m a_{ij}\lambda_i) > 0$, and $\bar{x}_j \in \{0,1\}$, if $(c_j - \sum_{i=1}^m a_{ij}\lambda_i) = 0$. Observe that, in general, the vector $\bar{x}$ is not a feasible solution of SC.

For any $\lambda \geqslant 0$, $L(\lambda)$ is a valid lower bound of SC. In fact, it can be shown that the optimum value of the Lagrangian function is equal to the value of the linear programming relaxation of SC. Moreover, the vector λ which maximizes the Lagrangian function can be found by means of a simple iterative procedure called the *subgradient algorithm*.

The elements of the vector λ are called the *Lagrange multipliers*, while the elements of $(c - A^{\mathrm{T}}\lambda)$ are called the *reduced costs*.

If an upper bound z_{UB} to the value of an optimal solution of SC is available, we can compute a *gap*, $g = z_{\mathrm{UB}} - L(\lambda)$, which measures the quality of the relaxation.

Moreover, the value g can be used to discover variables which are at zero in any feasible solution of SC whose value is strictly lower than z_{UB}. In particular, a variable can be fixed at zero if its reduced cost is strictly greater than g (*reduced cost fixing*).

The two main ingredients of the success of Beasley's heuristic [11] are the reduced cost fixing and the idea of completing to a feasible solution of SC the vector $\bar{x}$ which minimizes $(c - A^T\lambda)x$ at each iteration of the subgradient algorithm. Unfortunately, both ideas are ineffective in the large-scale problems proposed by Ferrovie dello Stato. In fact, since the costs can be only 1 or 2, we have that as soon as the gap is greater than 2, which happens very often, the reduced cost fixing is inoperative. On the other hand, we also observed that the feasible solutions obtained by extending the solution $\bar{x}$, of the Lagrangian problem, are of very poor quality and their computation is very time consuming.

We now describe in more detail our approach. The algorithm is roughly divided into two phases. In the first phase, we work with the entire problem data. For the large-scale problems that we are dealing with, these impose severe restrictions on

Algorithm *Set_Covering_Heuristic*;

begin

{ *Phase 1* }

- Read and compress the problem data of (*SC*);
- Preprocess the constraint matrix by removing dominated columns and rows covered by a unique column;
- Apply the subgradient method to find a "good" vector of dual multipliers λ^* for the Lagrangian relaxation of (*SC*);
- Compute the reduced costs corresponding to λ^*;
- Select the core problem.

{ *Phase 2* }

repeat as many times as you like

begin

while the core problem is non-empty **do**

begin

- Solve the primal and dual Lagrangian relaxations;
- Select one variable to be fixed at 1;
- Perform the completion heuristic;
- Reduce the problem;

end

Restore the original core problem;

end

end

Fig. 1. Our algorithm.

time and space usage. Indeed, as we will see in Section 2.1, we read and store the problem data in a compressed form while performing column domination checks among adjacent columns. After that we compute, by means of an efficient implementation of the subgradient method applied to the Lagrangian relaxation of SC, a lower bound for the value of the optimal solution along with a vector of dual multipliers λ^* and the corresponding reduced costs. Finally, we construct a manageable size problem by selecting a subset of the columns of A. In particular, we select all the columns whose reduced costs are below a given threshold γ. We then add (in ascending reduced cost order) enough columns to ensure that every row is covered by at least *cover_num* columns (if possible). In our computational experiments, we set $\gamma = 0.1$ and *cover_num* $= 10$. The problem whose constraint matrix is obtained in this way is called the *core problem*.

The goal of the second phase of the algorithm is to produce an acceptably good feasible solution for the core problem. Its basic component is a greedy heuristic which, at each step, selects an index of a column to be added to the current partial (0-1) solution, i.e. the corresponding variable is fixed to 1. The column is selected based on the information obtained from a primal-dual Lagrangian heuristic described in Section 2.2. The size of the problem is then reduced in a reduction step to eliminate redundant columns. The greedy heuristic terminates when the core problem becomes empty.

At each step of the greedy heuristic, we use the information obtained from the Lagrangian procedure to complete the current partial (0-1) solution to a feasible solution of SC. This solution provides an upper bound that we use in the reduction step. Evidently, the quality of this solution is not as good as that of the solution obtained by iteratively fixing a variable, reducing the problem and recomputing the optimal Lagrangian multipliers. Nevertheless, its quality improves as the algorithm proceeds.

The greedy heuristic is repeated several times. The first time the variable selection is performed by means of a deterministic rule. In all the subsequent executions, the greedy algorithm chooses, at each step, among the two best candidate variables by "tossing a coin".

2.1. Compressing the initial data

Because of the nature of pairing generation algorithms, that in most cases proceed by "enumerative" techniques, the columns of the set-covering formulation of the crew-scheduling problem have a very particular structure. Namely, it is frequently the case that adjacent columns are very similar to each other. This points to the need of using more efficient data structures that exploit this property.

Since the constraint matrix is very sparse (and large-scale), it is necessary to use a sparse representation of the non-zero elements. In practice, this is done by storing the matrix in major-column form (for each column of the matrix the non-zero elements are written consecutively), with another vector of pointers to the start of every

Algorithm *Primal_Dual_Lagrangian* $(\lambda, \mu, z^*_{LB}, z^*_{UB})$;

begin

$pf := df := f_0$;

while $((pf > f_{min})$ **or** $(df > f_{min}))$ **and** $((z^*_{UB} - z^*_{LB}) > \Delta)$ **do**

begin

{ *Compute primal and dual reduced costs* }

for $j := 1$ to n **do**

$red\_cost_j := c_j - \sum_{i=1}^{m} a_{ij}\lambda_i$;

if $red\_cost_j < 0$ **then** $x_j := 1$ **else** $x_j := 0$;

for $i := 1$ to m **do**

$d\_red\_cost_i := 1 - \sum_{j=1}^{n} a_{ij}\mu_j$;

if $d\_red\_cost_i > 0$ **then** $y_i := \bar{c}_i$ **else** $y_i := 0$;

{ *Compute primal and dual slacks* }

for $i := 1$ to m **do** $slack_i := 1 - \sum_{j=1}^{n} a_{ij}x_j$;

for $j := 1$ to n **do** $d\_slack_j := \sum_{i=1}^{m} a_{ij}y_i - c_j$;

{ *Update upper and lower bounds* }

$z_{UB} := \sum_{j=1}^{n} c_j\mu_j + \sum_{i=1}^{m} y_i \cdot d\_red\_cost_i$; $z_{LB} := \sum_{i=1}^{m} \lambda_i + \sum_{j=1}^{n} x_j \cdot red\_cost_j$;

$z^*_{UB} := \min\{z^*_{UB}, z_{UB}\}$; $z^*_{LB} := \min\{z^*_{LB}, z_{LB}\}$;

{ *Compute primal and dual subgradient steps* }

$\sigma := \sum_{i=1}^{m} slack_i^2$;

if $\sigma > 0$ **then** $p\_step := pf(z^*_{UB} - z_{LB})/\sigma$ **else** $p\_step := 0$;

$d\_\sigma := \sum_{j=1}^{n} d\_slack_j^2$;

if $d\_\sigma > 0$ **then** $d\_step := df(z_{UB} - z^*_{LB})/d\_\sigma$ **else** $d\_step := 0$;

{ *Update primal and dual variables* }

for $i := 1$ to m **do** $\lambda_i := \max(0, \lambda_i + p\_step \cdot slack_i)$;

for $j := 1$ to n **do** $\mu_j := \max(0, \mu_j + d\_step \cdot d\_slack_j)$;

{ *Update the step adjustment parameters* }

if in the last β iterations z^*_{LB} has not been updated **then** $pf := pf/\rho$;

if in the last β iterations z^*_{UB} has not been updated **then** $df := df/\rho$;

end

end

Fig. 2. A primal-dual subgradient algorithm.

column, and a third vector that for every non-zero element, gives the index of the corresponding row of the constraint matrix. Since in a set-covering problem all the non-zero elements are 1, the first vector is not needed. Moreover, in the data provided to us, we observed that adjacent columns differ only by few elements. Hence, we store the constraint matrix by difference. Namely, for every column j we store with positive sign the indices of the rows belonging to column j but not to column $j-1$ and with negative sign the indices of the rows belonging to column $j-1$ but not to column j. Notice that whenever the entries of a column j are all positive or all negative, a possible domination between adjacent columns is detected. By using this representation, we could save approximately 50% storage space.

It is possible to derive an efficient implementation of the subgradient method for the optimization of $L(\lambda)$ that exploits this data structure. The optimization process can also be improved by applying a simple preprocessing technique to eliminate redundant columns. A full preprocessing of the constraint matrix involves a computational effort of roughly the square of the number of columns of the constraint matrix, which is computationally infeasible for large-scale problems. On the other hand, it is rarely the case, because of the way that columns are generated, that similar or dominated columns appear in very different parts of the constraint matrix. We found to be a right balance to just compare adjacent columns for domination or duplication. Typically, this reduced the number of columns in the matrix by 10%.

2.2. A primal-dual subgradient algorithm

One of the main differences of the algorithm we are describing with respect to other approaches in the literature that also use Lagrangian relaxation, is that we also use the following Lagrangian relaxation of DLSC, namely,

$$\min \ \mathrm{DL}(\mu)$$

where $\mu \in \mathbb{R}^n_+$ and

$$\mathrm{DL}(\mu) = \max \ (\ \mathbf{1} - A\mu)y + c\mu$$

with the maximum taken over $0 \leqslant y \leqslant \bar{c}$, where $\bar{c}_i = \min\{c_j\colon\ j \text{ such that } a_{ij} = 1\}$. Observe that the constraint $y_i \leqslant \bar{c}_i$, although not explicitly stated in the formulation DLSC, is implied by the constraints $A^{\mathrm{T}}y \leqslant c$.

Given μ, the value of $\mathrm{DL}(\mu)$ can be easily computed by studying the signs of the components of $(\ \mathbf{1} - A\mu)$. In fact, it is easily seen that any vector $\bar{y}$, which maximizes $(\mathbf{1} - A\mu)y$, is obtained by setting $\bar{y}_i = \bar{c}_i$, if $(1 - \sum_{j=1}^n a_{ij}\mu_j) > 0$, $\bar{y}_i = 0$, if $(1 - \sum_{j=1}^n a_{ij}\mu_j) < 0$, and $\bar{y}_i \in [0, \bar{c}_i]$, if $(1 - \sum_{j=1}^n a_{ij}\mu_j) = 0$. For any $\mu \geqslant 0$, $\mathrm{DL}(\mu)$ gives a valid upper bound of LSC. The elements of $(\ \mathbf{1} - A\mu)$ are called the *dual reduced-costs*.

An interesting observation is the fact that DLSC has a very similar structure to LSC, and hence similar convergence properties of the subgradient algorithm used

to solve their Lagrangian relaxations can be expected. This is particularly true in the case where the costs $c_i \in \{1,2\}$, as in the data provided to us by Ferrovie dello Stato. We now present a primal-dual subgradient Lagrangian algorithm that solves the Lagrangian relaxations of LSC and DLSC simultaneously. A first advantage of such a method lies in the fact that for any $\bar{\lambda} \in \mathbb{R}^m_+$ and $\mu \in \mathbb{R}^n_+$, the values $L(\bar{\lambda})$ and $\mathrm{DL}(\bar{\mu})$ constitute valid lower and upper bounds, respectively, for the value of the linear programming relaxation of SC. This allows a better estimate of the error at each iteration of the subgradient method, and hence improves the overall convergence properties of the algorithm. Moreover, by solving both Lagrangian functions, primal and dual information about LSC can be obtained, which is useful for the variable selection step. In what follows we give a brief description of the algorithm.

We used an efficient implementation of the primal-dual algorithm which incrementally updates all the relevant parameters (reduced costs, slacks and multipliers, both primal and dual), rather than computing them from scratch at each iteration. To be more specific, at each iteration we compute the vectors Δ_λ and Δ_μ which represent the difference between the old and the new multipliers. Such vectors are much less dense than the vectors of multipliers themselves. We take advantage of the sparsity of such vectors while updating the reduced costs. In fact, we scan the rows corresponding to the non-zero components of Δ_λ to update the primal reduced costs. In a similar way we use the vector Δ_μ to update the dual reduced costs. Subsequently, for each column, we compare the old and new reduced costs. If we detect a change from non-negative to negative (or vice versa), we update (adding or subtracting one) the slacks of all the rows having a non-zero coefficient in that column. We deal similarly with the updating of the dual slacks. Finally, the slacks which have become non-zero are used to compute the vectors Δ_λ and Δ_μ for the next iteration. This incremental version of the algorithm is much faster than the naive one described in detail.

As usual, the primal (dual) subgradient step size is controlled by a parameter *pf* (*df*). Both *pf* and *df* are initially set to the value f_0, and the algorithm terminates when both *pf* and *df* fall below the value $f_{\min}$.

Evidently, the choice of the values of parameters affects both the accuracy and the execution time of the procedure. Namely, low values for ρ, $f_{\min}$ and Δ, and high values for f_0 and β, guarantee an accurate convergence to the optimum of the linear programming relaxation of SC, but cause a significant increase in the computation time. To compromise among the two requirements of accuracy and fast execution time we have decided to choose $\rho = 1.2$, $f_{\min} = 0.002$, $\Delta = 0.01$, $f_0 = 4$ and $\beta = 15$, for the first execution of the primal-dual subgradient procedure in Phase 2, and $\rho = 2$, $f_{\min} = 0.02$, $\Delta = 0.1$, $f_0 = 2$ and $\beta = 5$, for subsequent executions.

As we remarked at the beginning of this section, unlike other large-scale implementations of Lagrangian heuristics for the set-covering problem [11], the primal-dual algorithm does not attempt to perform reduced cost fixing of variables at each iteration of the subgradient algorithm. On the contrary, we do that only when the

subgradient algorithm terminates (by using the optimal vector of Lagrange multipliers) and the size of the reduced problem is sufficiently small (e.g.the number of rows is $\frac{1}{8}$ of the original number).

2.3. Variable selection

We now focus on the selection of a variable to be fixed to one in each step of the greedy algorithm. As observed at the beginning of Section 2 this amounts to specifying the merit function used to rank the variables. A well-known heuristic for obtaining good solutions of general integer programming problems is to fix to 1 the variable which is closer to this value in the solution of the linear programming relaxation of the problem. Unfortunately, solving the linear relaxation of the core problem is too time consuming for large-scale instances and hence computationally prohibitive as a step of the greedy heuristic.

For this reason we resorted to use the optimal Lagrange multipliers $\bar{\mu}$ obtained by the primal-dual algorithm described in Section 2.2, which constitutes a near-optimal and near-feasible solution of the linear relaxation. In fact, observe that if $1 - \sum_{j=1}^{n} a_{ij}\bar{\mu}_j \leqslant 0$, for each i, then $\bar{\mu}$ is a feasible (and near-optimal) fractional solution of LSC. In general, not all dual reduced costs are non-positive. Nevertheless, they are usually very small and so the information provided by $\bar{\mu}$ proves to be effective in practice.

Another useful information provided by the primal-dual Lagrangian heuristic is the vector of primal reduced costs. Indeed, a heuristic rule for the variable selection step could be that of fixing at 1 a variable which causes the minimum increase in the value of the linear relaxation of the resulting subproblem. As observed in discussing the reduced cost fixing, a variable with positive reduced cost causes, when fixed at one, an increase of the lower bound by an amount at least equal to the value of its reduced cost. Hence, it seems reasonable to fix at one the variable with minimum reduced cost.

In our procedure we combine the two above mentioned criteria. In particular, we choose the variable x_j which maximizes the merit function $\mu_j - \alpha \cdot red\_cost_j$. The parameter α is set to 12 in the first execution of the greedy algorithm and chosen in turn from the set $\{4, 8, 12\}$ in the subsequent iterations. In later iterations we choose with probability $1/2$ the second best variable instead of the first one.

In every step, we also fix at 1 all the variables x_j with $\mu_j \geqslant 0.99$ and $red\_cost_j \leqslant 0.01$, and at 0 all the variables whose reduced cost is greater than the gap (reduced cost fixing).

After fixing the variables we reduce the problem by removing the corresponding columns and all the rows having 1 in a column whose variable has been fixed at 1. After this reduction, we compare adjacent columns for domination or duplication and remove the redundant ones.

3. Computational experience

3.1. The test-bed

The test-bed we used consists of set-covering problems arising from crew-scheduling at the Italian railways. They have sizes ranging from 50,000 variables and 500 rows to 1,000,000 variables and 5000 rows. On average, they have approximately 10 non-zero entries per column. The instances correspond to real data from the Italian railroad network. They were provided to us as a part of the FASTER challenge, a competition among universities and research institutions in Italy to find efficient solutions to large-scale set-covering problems arising from this application. These data were also provided to several industrial competitors, who were bidding for a software and hardware contract with the company. The competition was organized as follows. First, for a period of approximately six months, every group was given a set of sample problems, RAIL516, RAIL507, RAIL2586, and RAIL4872, so as to test their respective codes. The problems were divided into three categories, small, medium and large. At the end of this test-period, all participants gave their codes to Ferrovie dello Stato. The competition was divided into two parts, one for small instances and the other for medium and large instances. The small problems were run on a PC 486/66, with 16 Mb of memory, and with a time limit of 3000 CPU seconds. The medium and large problems were run on an HP 735 with 256 Mb of memory and with a time limit of 10,000 CPU seconds. In each category they announced a winner (a group could not win in more than one category), the group that had obtained the solution with minimum total cost.

3.2. Computational results

In Table 1 we include the results for the test problems. The problem sizes (number of rows $\times$ number of columns), and categories (small (s), medium (m) and large (l)), are included in the column "Size". We also present information about the value of the linear programming relaxation "Lower Bound", the solution obtained by our algorithm "Upper Bound", and the best solution reported by the industry "Best Industry". For the medium and large problems our algorithm was run for 10,000 seconds on an IBM RS/6000 375 with 32 Mb of memory, and in many cases the solution reported was

Table 1
Test problems

Problem	Size	Lowerbound	Upper bound	Best industry
RAIL507	(507 $\times$ 63,009) (s)	172.4	174	174
RAIL582	(582 $\times$ 55,515) (s)	209.5	211	211
RAIL2586	(2586 $\times$ 920,683) (m)	936.1	951	952
RAIL4872	(4872 $\times$ 968,672) (l)	1509.0	1534	1538

found at the earlier stages of the procedure. The small problems were run for 1000 seconds on the same machine. Unfortunately, we do not have the exact running times for the solutions provided by the industry, but we were told by Ferrovie dello Stato that they exceeded the time limit set by us on a comparable computer.

Table 2 contains the results for the problems used in the competition. Our results are included under the column "Roma". We also present the values of the solutions obtained by the other participant groups, under the names of their respective universities.

The reader should notice that even though the competition problems have sizes which are very similar to the test problems, the value of the optimal solution is quite different (in fact, the lower bounds on the optimal values of the test problems are sensibly greater than the upper bounds obtained for the problems of similar size proposed in the competition). The reason for this disparity is that the competition problems were much denser than the ones used for the computational tests. No parameter adjustment was done on the day of the competition to account for this fact. All groups used their codes as submitted before the competition. Unfortunately, we do not precisely know what algorithms did the other groups use. In informal conversations we could establish that the group at Bologna used an algorithm which is very similar to ours, while the group at Trento used an approach based on Tabu search, and the group at Ancona a subgradient heuristic in the spirit of Beasley's [11]. Overall, our code was only second to the one of Bologna's, and quite effective when compared to the other codes, which are mostly based on approaches already in the literature.

All the data sets of the problems RAILxxx have been recently included in the collection of test problems called *OR-Library* which is maintained by Beasley and is accessible via FTP (graph.ms.ic.ac.uk/pub).

As a final experiment, we tested the robustness of our algorithm by solving 10 set-covering problems generated by Beasley which are considered to be the hardest randomly generated set-covering problems of the OR-Library collection. These problems have all 1000 rows and 10,000 columns and integer costs between 1 and 100. The problems *SCPNRGx* have density 2% while the problems *SCPNRHx* have density 5%. As it is promptly seen, the cost structure of these problems is quite different from that of the RAILxxx problems. In principle, this difference could affect the assumption we made in designing our algorithm. For example, performing the reduced cost fixing while computing the Lagrangian bound or not performing the "storage by dif-

Table 2
Competition problems

Problem	Size	Bologna	Roma	Trento	Ancona
RAIL516	$(516 \times 47,311)$ (s)	182	182	**	188
RAIL2536	$(2536 \times 1,081,841)$ (m)	691	692	709	745
RAIL4284	$(4284 \times 1,092,610)$ (l)	1065	1070	1117	1175

Table 3
Beasley's data set

Problem	Size	B-Lag	JB-Ann	BC-Gen	PD-Lag	Lower bounds
SCPNRG1	(1000 × 10,000)	184	179	176	176	159.5
SCPNRG2	(1000 × 10,000)	163	158	155	155	141.8
SCPNRG3	(1000 × 10,000)	174	169	166	167	147.7
SCPNRG4	(1000 × 10,000)	176	172	168	170	148.4
SCPNRG5	(1000 × 10,000)	175	168	168	169	147.8
SCPNRH1	(1000 × 10,000)	68	64	64	64	47.3
SCPNRH2	(1000 × 10,000)	66	64	64	64	47.7
SCPNRH3	(1000 × 10,000)	65	60	59	60	44.3
SCPNRH4	(1000 × 10,000)	63	59	59	59	43.3
SCPNRH5	(1000 × 10,000)	60	55	55	55	41.7

ference" could be, in this case, much more effective. Nonetheless, we did not modify our algorithm in solving these problems. Moreover, we found we could obtain very good results with the same values of the main parameters with the RAILxxx problems, with the minor exception of the parameter *cover_num* which was set to 60. This is certainly an indication that our algorithm is very robust, regardless of the special characteristics of the problems at hand.

The results of this experiment are presented in Table 3, where we compare our algorithm with results obtained by Beasley in [11], Jacobs and Brusco [13], and Beasley and Chu [14]. Our results are included under the column "PD-Lag", Beasley's results under "B-Lag", Jacobs and Brusco under "JB-Ann", Beasley and Chu under "BC-Gen". The last two are based on general purpose heuristics, the first one corresponds to a simulated annealing approach and the last one to a genetic algorithm. For this class of problems, our heuristic was run for a total of 1000 CPU seconds on an IBM RS/6000 375. The most effective heuristic (Beasley and Chu) was run 10 times on each problem, each run with an average running time of about 2500 CPU seconds on a Silicon Graphics Indigo Workstation. Our heuristic performed remarkably well on this data set. In six out of 10 cases, our results were as good as the best solution known up to date, and were obtained in smaller computational times.

Acknowledgements

We wish to thank Ing. P.L. Guida of Ferrovie dello Stato for involving us in the FASTER challenge and providing us with the data sets.

References

[1] R. Anbil, E. Gelman, B. Patty, R. Tanga, Recent advances in crew pairing optimization at American Airlines, Interfaces 21 (1991) 62–74.
[2] J. Barutt, T. Hull, Airline crew-scheduling: supercomputers and algorithms, SIAM News 23 (1990) 1–2.

[3] D.R. Bornemann, The evolution of airline crew pairing optimization, AGIFORS Crew Management Study Group Proceedings, Paris, 1982.
[4] I. Gershkoff, Optimizing flight crew schedules, Interfaces 19 (1989) 29–43.
[5] K.L. Hoffman, M. Padberg, Solving airline crew-scheduling problems by branch-and-cut, Management Science 39 (1993) 657–682.
[6] S. Lavoie, M. Minoux, E. Odier, A new approach for crew pairing problems by column generation with application to air transportation, European Journal of Operations Research 35 (1988) 45–58.
[7] V. Chvatal, A greedy heuristic for the set-covering problem, Mathematics of Operations Research 4 (1979) 233–235.
[8] E. Balas, A. Ho, Set covering algorithms using cutting planes, heuristics, and subgradient optimization: a computational study, Mathematical Programming Study 12 (1980) 37–60.
[9] T.A. Feo, M.G.C. Resende, A probabilistic heuristic for a computationally difficult set covering problem, Operations Research Letters 8 (1989) 67–71.
[10] M.L. Fisher, P. Kedia, Optimal solution of set covering/partitioning problems using dual heuristics, Management Science 36 (1990) 674–688.
[11] J.E. Beasley, A Lagrangean heuristic for set-covering problems, Naval Research Logistics 37 (1990) 151–164.
[12] E. Balas, M. Carrera, A dynamic subgradient-based branch-and-bound procedure for set covering, Management Science Research Report No. 568, Carnegie Mellon University, 1992, Operations Research (to appear).
[13] L.W. Jacobs, M.J. Brusco, A simulated annealing-based heuristic for the set-covering problem, Working Paper, Operations Management and Information System Department, Northern Illinois University, Dekalb, IL60115, USA, 1993.
[14] J.E. Beasley, P.C. Chu, A genetic algorithm for the set covering problem, Working Paper, The Management School, Imperial College, London SW7 2AZ, England, 1994.