A Heuristic Method for the Set Covering Problem

Alberto Caprara*, Matteo Fischetti• and Paolo Toth*

* DEIS, University of Bologna, Italy

• DMI, University of Udine, Italy

Abstract

We present a Lagrangian-based heuristic for the well-known *Set Covering Problem* (SCP). The algorithm was initially designed for solving very large scale SCP instances, involving up to 5,000 rows and 1,000,000 columns, arising from crew scheduling in the Italian Railway Company, Ferrovie dello Stato SpA. In 1994 Ferrovie dello Stato SpA, jointly with the Italian Operational Research Society, organized a competition, called FASTER, intended to promote the development of algorithms capable of producing good solutions for these instances, since the classical approaches meet with considerable difficulties in tackling them. The main characteristics of the algorithm we propose are (1) a dynamic pricing scheme for the variables, akin to that used for solving large-scale LP's, to be coupled with subgradient optimization and greedy algorithms, and (2) the systematic use of column fixing to obtain improved solutions. Moreover, we propose a number of improvements on the standard way of defining the step-size and the ascent direction within the subgradient optimization procedure, and the scores within the greedy algorithms. Finally, an effective refining procedure is proposed. Our code won the first prize in the FASTER competition, giving the best solution value for all the proposed instances. The algorithm was also tested on the test instances from the literature: in 92 out of the 94 instances in our test bed we found, within short computing time, the optimal (or the best known) solution. Moreover, among the 18 instances for which the optimum is not known, in 6 cases our solution is better than any other solution found by previous techniques.

The *Set Covering Problem* (SCP) is a main model for several important applications, including *crew scheduling*, where a given set of *trips* has to be covered by a minimum-cost set of *pairings*, a pairing being a sequence of trips that can be performed by a single crew. A widely-used approach to crew scheduling works as follows. First, a very large number of pairings are generated. Then an SCP is solved, having as row set the trips to be covered, and as column set the pairings generated. In railway applications, very large scale SCP instances typically arise, involving thousands of rows and millions of columns. The classical methods proposed for SCP meet with considerable difficulties in tackling these

instances, concerning both the computing time and the quality of the solutions found. On the other hand, obtaining high-quality solutions can result in considerable savings. For this reason, in 1994 the Italian railway company, *Ferrovie dello Stato SpA*, jointly with the Italian Operational Research Society, *AIRO*, decided to organize a competition called FASTER (Ferrovie Airo Set covering TendER) among the departments of Italian universities, possibly including foreign researchers. Some well-known researchers from all over the world took part in the competition.

Three size classes of SCP instances were defined for FASTER, namely a *micro* size class, with up to 600 rows and 60,000 columns, a *medium* size class, with up to 2,750 rows and 1,100,000 columns, and a *large* size class, with up to 5,500 rows and 1,100,000 columns. Each participant had to implement an algorithm, and send the corresponding code to Ferrovie dello Stato SpA. A prize was to be assigned, for each size class, to the code giving, for a particular instance, the best solution value within a given time limit. The three prizes were mutually exclusive, and amounted to approximately US $ 60,000 for the large instance, US $ 30,000 for the medium instance, and US $ 15,000 for the micro instance. The time limit was 3,000 seconds on a PC 486/33 with 4 Mb RAM for the micro instance, and 10,000 seconds on a HP 735/125 workstation with 256 Mb RAM for the medium and large instances. Four sample instances of different sizes were distributed before the competition. The SCP packages available on the market were tried by Ferrovie dello Stato SpA on these test instances. Of course, the instances to be solved in the competition were not distributed in advance.

We took part in the FASTER competition as Dipartimento di Elettronica, Informatica e Sistemistica, University of Bologna. Our code ranked first for all the three size-class instances. We also obtained the best solution values on all sample instances. For the medium and large sample instances, our solutions were strictly better than those provided by the commercial packages available on the market. After the competition, we tested our code on a wide set of SCP instances from the literature, with remarkably good results: in 92 out of the 94 instances in our test bed we found, within short computing time, the optimal (or the best known) solution. Moreover, among the 18 instances for which the optimum is not known, in 6 cases our solution is better than any other solution found by previous techniques.

The main characteristics of the algorithm we propose are (1) a dynamic pricing scheme for the variables, akin to that used for solving large-scale LP's, to be coupled with subgradient optimization and greedy algorithms, and (2) the systematic use of column fixing to obtain improved solutions. Moreover, we propose a number of improvements on the standard way of defining the step-size and the ascent direction within the subgradient optimization procedure, and the scores within the greedy algorithms. Finally, an effective

refining procedure is proposed.

The paper is organized as follows. Section 1 gives an overview of the overall algorithm, which mainly consists of three parts: a subgradient phase, a heuristic phase, and a column fixing phase dealt with, respectively, in Sections 2, 3, and 4. Sections 5 and 6 describe the pricing technique and the refining procedure. The overall method is summarized in Section 7, and finally extensive computational experiments are reported in Section 8.

1 General Framework

SCP can be formally defined as follows. Let $A = (a_{ij})$ be a 0-1 $m \times n$ matrix, and $c = (c_j)$ be an n-dimensional integer vector. In the following we refer to the rows and columns of A simply as rows and columns. Let $M = \{1, \ldots, m\}$ and $N = \{1, \ldots, n\}$. The value c_j $(j \in N)$ represents the cost of column j, and we assume without loss of generality $c_j > 0$ for $j \in N$. We say that a column $j \in N$ *covers* a row $i \in M$ if $a_{ij} = 1$. SCP calls for a minimum-cost subset $S \subseteq N$ of columns, such that each row $i \in M$ is covered by at least one column $j \in S$. A mathematical model for SCP is

$$v(\text{SCP}) = \min \sum_{j \in N} c_j x_j \tag{1}$$

subject to

$$\sum_{j \in N} a_{ij} x_j \geq 1 \quad i \in M \tag{2}$$

$$x_j \in \{0, 1\} \quad j \in N \tag{3}$$

where $x_j = 1$ if $j \in S$, $x_j = 0$ otherwise. For notational convenience, for each row $i \in M$ let

$$J_i = \{j \in N : a_{ij} = 1\}$$

be the set of columns covering row i. Analogously, for each column $j \in N$ let

$$I_j = \{i \in M : a_{ij} = 1\}$$

be the row subset covered by column j. Moreover, let $q = \sum_{i \in M} \sum_{j \in N} a_{ij}$ denote the number of nonzero entries of A.

SCP is NP-hard in the strong sense, and has many practical applications, see e.g. Balas (1983). Many algorithms have been proposed in the literature for the exact solution of the problem, see Balas and Ho (1980), Beasley (1987), Fisher and Kedia (1990), Beasley and Jörnsten (1992), Nobili and Sassano (1992), and Balas and Carrera (1996). These exact algorithms can solve instances with up to a few hundred rows and a few thousand columns. When larger scale SCP instances are tackled, heuristic algorithms

are needed. Classical *greedy* algorithms are very fast in practice, but typically do not provide high quality solutions, as reported in Balas and Ho (1980) and Balas and Carrera (1996). The most effective heuristic approaches to SCP are those based on *Lagrangian relaxation* with *subgradient optimization*, following the seminal work by Balas and Ho (1980), and then the improvements by Beasley (1990), Fisher and Kedia (1990), Balas and Carrera (1996), and Ceria, Nobili and Sassano (1995). Lorena and Lopes (1994) propose an analogous approach based on *surrogate relaxation*. Wedelin (1995) proposes a general heuristic algorithm for integer programs having a 0-1 constraint matrix; the algorithm is based on Lagrangian relaxation with coordinate search, where a suitably-defined approximation term is introduced. Recently, Beasley and Chu (1996) proposed an effective genetic algorithm.

Our heuristic scheme is based on dual information associated with the widely-used Lagrangian relaxation of model (1)-(3). We assume the reader is familiar with Lagrangian relaxation theory; see, e.g., Fisher (1981) for an introduction. For every vector $u \in R_+^m$ of Lagrangian multipliers associated with the constraints (2), the Lagrangian subproblem reads:

$$L(u) = \min \sum_{j \in N} c_j(u) x_j + \sum_{i \in M} u_i \tag{4}$$

subject to

$$x_j \in \{0,1\} \quad j \in N \tag{5}$$

where $c_j(u) = c_j - \sum_{i \in I_j} u_i$ is the *Lagrangian cost* associated with column $j \in N$. Clearly, an optimal solution to (4)-(5) is given by $x_j(u) = 1$ if $c_j(u) < 0$, $x_j(u) = 0$ if $c_j(u) > 0$, and $x_j(u) \in \{0,1\}$ when $c_j(u) = 0$. The Lagrangian dual problem associated with (4)-(5) consists of finding a Lagrangian multiplier vector $u^* \in R_+^m$ which maximizes the lower bound $L(u)$. As (4)-(5) has the *integrality property*, any optimal solution u^* to the dual of the *Linear Programming* (LP) relaxation of SCP, namely problem $\max \left\{ \sum_{i \in M} u_i : \sum_{i \in I_j} u_i \leq c_j \ (j \in N), u_i \geq 0 \ (i \in M) \right\}$, is also an optimal solution to the Lagrangian problem; see Fisher (1981). On the other hand, computing an optimal multiplier vector by solving an LP is typically time-consuming for very large scale instances. A commonly used approach for finding near-optimal multiplier vectors within a short computing time, uses the *subgradient vector* $s(u) \in R^m$, associated with a given u, defined by:

$$s_i(u) = 1 - \sum_{j \in J_i} x_j(u), \quad i \in M. \tag{6}$$

The approach generates a sequence $u^0, u^1, \ldots$ of nonnegative Lagrangian multiplier vectors, where u^0 is defined arbitrarily. As to the definition of u^k, $k \geq 1$, a possible choice

(Held and Karp, 1971) consists of using the following simple updating formula:

$$u_i^{k+1} = \max\left\{u_i^k + \lambda \frac{UB - L(u^k)}{||s(u^k)||^2} s_i(u^k), 0\right\} \qquad \text{for } i \in M, \tag{7}$$

where UB is an upper bound on $v(\text{SCP})$, and $\lambda > 0$ is a given *step size parameter*.

For near-optimal Lagrangian multipliers u_i, the Lagrangian cost $c_j(u)$ gives reliable information on the overall utility of selecting column j. Based on this property, we use Lagrangian (rather than original) costs to compute, for each $j \in N$, a *score* σ_j ranking the columns according to their likelihood to be selected in an optimal solution. These scores are given on input to a simple heuristic procedure, that finds in a greedy way a hopefully good SCP solution. Computational experience shows that almost equivalent near-optimal Lagrangian multipliers can produce SCP solutions of substantially different quality. In addition, no strict correlation exists between the lower bound value $L(u)$ and the quality of the SCP solution found. Therefore it is worthwhile applying the heuristic procedure for several near-optimal Lagrangian multiplier vectors.

Our approach consists of three main phases, described in detail in Sections 2 to 4. The first one is referred to as the *subgradient phase*. It is aimed at quickly finding a near-optimal Lagrangian multiplier vector, by means of an aggressive policy. The second one is the *heuristic phase*, in which a sequence of near-optimal Lagrangian vectors is determined and, for each vector, the associated scores are given on input to the heuristic procedure to possibly update the incumbent best SCP solution. In the third phase, called *column fixing*, we select a subset of columns having an estimated high probability of being in an optimal solution, and fix to 1 the corresponding variables. In this way we obtain an SCP instance with a reduced number of columns and rows, on which the three-phase procedure is iterated. According to our experience, column fixing is of fundamental importance to obtain high quality SCP solutions. The overall 3-phase heuristic is outlined next.

procedure 3-PHASE(x^*);
begin
    repeat
1. SUBGRADIENT PHASE:
    find a near-optimal multiplier vector u^*;
2. HEURISTIC PHASE:
    starting from u^*, generate a sequence of near-optimal multiplier vectors, and for each of them compute a heuristic solution to SCP (the best incumbent solution x^* being updated each time a better solution is found);
3. COLUMN FIXING:

select a subset of "good" columns and fix to 1 the corresponding variables

until x^* cannot be improved

end.

The repeat-until loop terminates whenever either all the rows are covered by the fixed columns, or, as almost always occurs, the sum of the costs of the fixed columns plus a lower bound on the cost of the residual problem is not less than the value of x^*.

When very large instances are tackled, the computing time spent on the first two phases becomes very large. We overcome this difficulty by defining a *core problem*, obtained from the original problem by keeping only a subset of the variables (columns), the remaining ones being fixed to 0. The choice of the columns in the core problem is often very critical, since an optimal solution typically contains some columns that, although individually worse than others, must be selected in order to produce an overall good solution. Hence we decided not to "freeze" the core problem. Instead, we use a *variable pricing* scheme to update the core problem iteratively (see Section 5), in a vein similar to that used for solving large size LP's.

After each application of procedure 3-PHASE, a *refining procedure* is used (see Section 6), which in some cases produces improved solutions.

In the following, M will denote the set of the rows that are not covered by the currently fixed columns, and N the set of the columns covering at least one row in M.

2 Subgradient Phase

As already mentioned, this phase is intended to quickly produce a near-optimal Lagrangian multiplier vector. We use the updating formula (7).

The starting vector u^0 is defined in one of two different ways. In the first application of procedure 3-PHASE, u^0 is defined in a greedy way as follows:

$$u_i^0 = \min_{j \in J_i} \frac{c_j}{|I_j|}, \qquad i \in M. \tag{8}$$

As to the other applications of procedure 3-PHASE, we start from the best multiplier vector (i.e., the one producing the best lower bound for the subproblem defined by the rows in M), say u^*, computed before the last column fixing, and obtain the starting multiplier vector u^0 through random perturbation of u^*. To be more specific, we first remove from u^* all the entries associated with the rows covered by the columns fixed in the last iteration, and define $u_i^0 = (1 + \delta_i)u_i^*$ for the remaining rows i, where δ_i is a uniformly random value in the range $[-0.1, 0.1]$. The perturbation lets the subgradient

phase converge to a different multiplier vector, hence it allows the subsequent heuristic phase to produce different, and hopefully better, SCP solutions.

The upper bound UB is set to the value of the best SCP solution found. As for its initial value, it is computed by applying the greedy heuristic in Section 3, by considering $u = 0$ (i.e., the original costs instead of the Lagrangian costs).

Parameter λ controls the step-size along the subgradient direction $s(u^k)$. The classical Held-Karp approach halves parameter λ if for p consecutive iterations no lower bound improvement occurs. In order to obtain a faster convergence, we have implemented the following alternative strategy. We start with $\lambda = 0.1$. Every $p = 20$ subgradient iterations we compare the best and worst lower bounds computed on the last p iterations. If these two values differ by more than 1%, the current value of λ is halved. If, on the other hand, the two values are within 0.1% from each other, we multiply the current value of λ by 1.5. This last choice is motivated by the observation that either the current u^k is almost optimal (in which case we are not interested, in this phase, in obtaining a slightly better multiplier vector), or the small lower bound difference is due to an excessively small step-size (that we contrast by increasing λ). According to our computational experience, this strategy leads to a faster convergence to near optimal multipliers, compared with the classical approach. Figure 1 compares the behavior of the two strategies applied to the instance RAIL516 described in Section 8.1 (both strategies use the improved subgradient direction discussed in the sequel). The classical approach halves λ, for the first time, at iteration 181, although in the previous iterations the lower bound growth is far from regular. On the other hand, choosing a smaller initial value for λ produces even worse results, in that the asymptotic convergence value is much worse. The new approach, however, recognizes a high variation of the lower bound in the early iterations, and reduces the step size. Our approach, which increases the value of λ under certain conditions, guarantees an overall robust procedure.

According to our computational experience, in many cases a large number of columns happen to have a Lagrangian cost $c_j(u)$ very close to zero. In particular, this occurs for large scale instances with costs c_j belonging to a small range, after a few subgradient iterations. For example, for instance RAIL516 more than 1000 Lagrangian costs with $|c_j(u)| < 0.001$ arise, on average, after each subgradient iteration. In this situation, the Lagrangian problem has a huge number of almost optimal solutions, each obtained by choosing a different subset of the almost zero Lagrangian cost columns. As a result, a huge number of subgradients $s(u^k)$ to be used in (7) exist. It is known that the steepest ascent direction is given by the minimum-norm convex combination of the above active subgradients. However, the exact determination of this combination is very time consuming, as it requires the solution of a quadratic problem. On the other hand, a

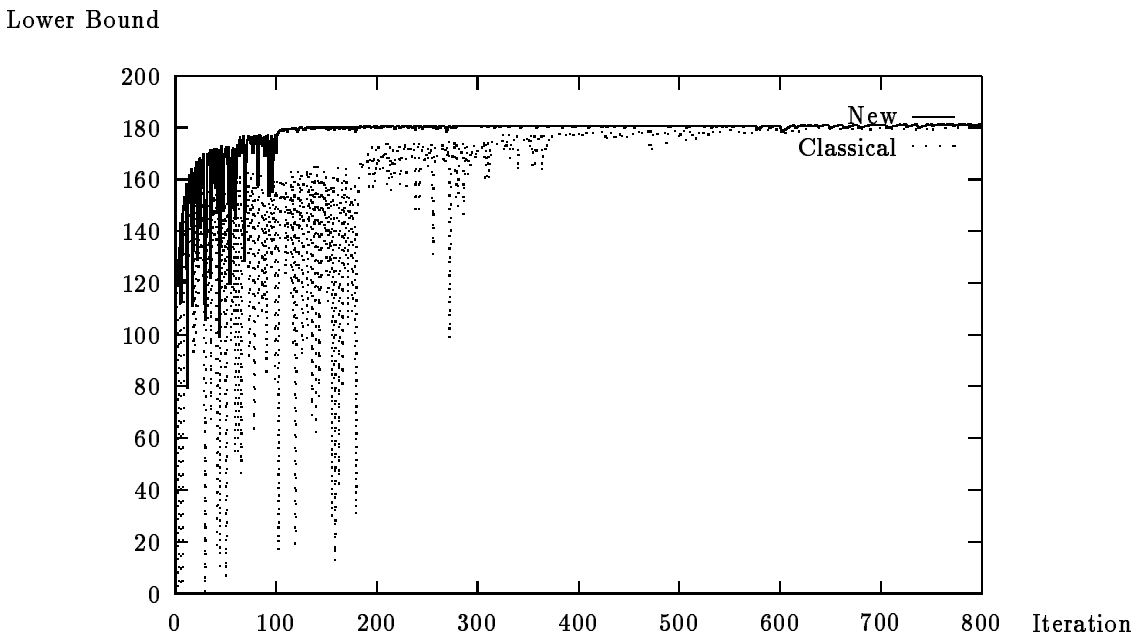

Figure 1: Comparison between the classical and new updating strategies for the step-size parameter λ on instance RAIL516.

random choice of the subgradient direction may produce very slow convergence due to zig-zagging phenomena. We overcome this drawback by heuristically selecting a small-norm subgradient direction, computed as follows. We first define a column subset $S = \{j \in N : c_j(u) \leq 0.001\}$. This set corresponds to a partial solution covering a row subset $I(S) = \bigcup_{j \in S} I_j$. Starting with S, we obtain a "prime" (i.e., minimal) partial cover by iteratively removing from S a *redundant column*, i.e., a column $j \in S$ such that $I(S \setminus \{j\}) = I(S)$. To be more specific, let $R \subseteq S$ be the set of the redundant columns of the initial S. We sort the columns in R according to decreasing costs $c_j(u)$. Then, for each $j \in R$, in the given order, we set $S = S \setminus \{j\}$ if $I(S) = I(S \setminus \{j\})$. Finally, we define $x_j(u) = 1$ for $j \in S$, $x_j(u) = 0$ for $j \in N \setminus S$, and compute the reduced norm "subgradient" $s(u)$ by means of (6) (notice that $s(u)$ is no longer guaranteed to be a subgradient). This phase requires $O(n \log n)$ time for column sorting, plus $O(q)$ time for the remaining computation. Computational results show that this choice leads to a faster convergence of the subgradient procedure. As an example, Figure 2 compares the behavior of the classical and new definitions of the subgradient directions to be used in (7), when applied to instance RAIL516 (both strategies use the improved updating rule for the step-size parameter λ, as described previously).

The subgradient phase ends as soon as we estimate the procedure converged to a near-optimal Lagrangian vector. This occurs when the lower bound improvement obtained in the last 300 subgradient iterations is smaller than 1.0, and, in percentage, below 0.1%. Our computational experience showed that the number of iterations needed to

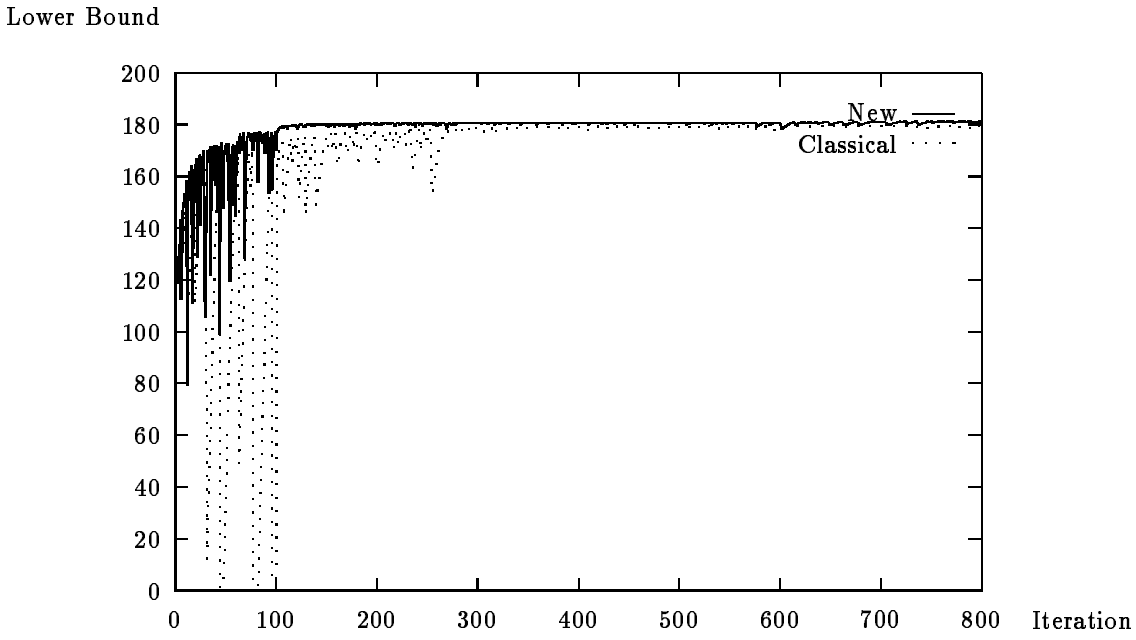

Figure 2: Comparison between the classical and new definitions of the subgradient directions on instance RAIL516.

reach convergence grows linearly with m. In any case, no more than $10m$ iterations are allowed (this limit was never reached in our computational experiments).

Each iteration of the subgradient phase requires (a) computing the Lagrangian costs associated with the current multiplier vector, which is done in $O(q)$ time; (b) computing the reduced norm "subgradient", in $O(n \log n + q)$ time; and (c) updating the multiplier vector, in $O(m)$ time. Since the maximum number of subgradient iterations allowed is $O(m)$, the overall time complexity of this phase is $O(m(n \log n + q))$.

3 Heuristic Phase

Let u^* be the best Lagrangian vector found during the subgradient phase. Starting with u^* we generate a sequence of Lagrangian vectors u^k in an attempt to "explore" a neighborhood of near-optimal multipliers. To this end, we update the multipliers as in the subgradient phase, but we do not reduce the subgradient norm, so as to allow for a change in a larger number of components of u^k. The heuristic phase ends after 250 iterations.

For each u^k, we apply the following greedy heuristic procedure to produce a "good" SCP solution S.

procedure GREEDY(u^k,S);
begin
1. Initialize $M^* := M$ to be the set of the currently uncovered rows,
and $S := \emptyset$ to be the set of the currently selected columns;
repeat
2. compute the score $\sigma_j := \text{SCORE}(j, u^k, M^*)$ for each $j \in N \setminus S$,
and let $j^* \in N \setminus S$ be a column with minimum score;
3. $S := S \cup \{j^*\}$; $M^* := M^* \setminus I_{j^*}$
until $M^* = \emptyset$
end.

The key step of the procedure is Step 2, in which the column scores σ_j are defined through function SCORE(j, u^k, M^*). Several rules have been proposed in the literature (see Balas and Ho, 1980, and Balas and Carrera, 1996) which define σ_j as a function of c_j and $\mu_j = |I_j \cap M^*|$ (e.g., $\sigma_j = c_j$, or $\sigma_j = c_j/\mu_j$). According to our computational experience, these rules produce good results when c_j is replaced by

$$\gamma_j = c_j - \sum_{i \in I_j \cap M^*} u_i^k,$$

since this term takes into account the dual information associated with the still uncovered rows M^*. The use of γ_j instead of c_j in a greedy-type heuristic was first proposed by Fisher and Kedia (1990).

We have also considered the following new rules, giving priority to columns having low cost γ_j and covering a large number μ_j of uncovered rows.

Rule a) $\sigma_j = \gamma_j/\mu_j$ if $\gamma_j > 0$, $\sigma_j = \gamma_j \mu_j$ if $\gamma_j < 0$.

Rule b) $\sigma_j = \gamma_j - \alpha\mu_j$ (where $\alpha > 0$ is a given parameter).

In all cases we set $\sigma_j = \infty$ whenever $\mu_j = 0$.

An extensive computational analysis showed that Rule a) outperforms Rule b) (tested with $\alpha = 0.1$, $\alpha = 0.01$, and $\alpha = 0.001$) and all other rules, when our heuristic scheme is applied. Indeed, we ran different versions of our code in which, for each given u^k, GREEDY(u^k, S) is applied by considering different rules for the definition of the scores. For all the instances we tried, the version using only Rule a) produced the same result as the one in which each call of GREEDY is replaced by a series of calls, one for each of the above-mentioned rules, and the best solution is chosen (the latter scheme required of course considerably more computing time). The same does not hold, however, when any rule alternative to a) is used. In view of this behavior, we decided to only use Rule a) within our final algorithm.

Observe that the values γ_j and μ_j need not be computed from scratch at each repeat-until iteration, as they can easily be updated before Step 3, by the following additional step:

2'. **for each** $i \in I_{j^*} \cap M^*$ **do**
for each $j \in J_i$ **do** let $\gamma_j := \gamma_j + u_i$ and $\mu_j := \mu_j - 1$;

In this way, the overall time complexity for the definition and the updating of γ_j and μ_j is $O(q)$ if an appropriate data structure is used (see Section 8), whereas the time required for each execution of Step 2 is $O(n)$. Therefore procedure GREEDY requires $O(rn + q)$ time, where $r \leq m$ is the cardinality of the solution found.

For the instances in our test bed (see Section 8) the term rn is typically much larger than q. To decrease the average computing time required by Step 2 we use the following approach. We determine the set $B \subseteq N \setminus S$ containing the $\min\{m, |N \setminus S|\}$ columns with smallest scores, and set τ to the current value of the smallest score of a column not in B. Whereas in Step 2' the values of γ_j and μ_j are updated for all columns $j \in N \setminus S$, in Step 2, as long as $\sigma_{j^*} \leq \tau$, the computation of σ_j and the search for j^* are performed only among the columns in B. When $\sigma_{j^*} > \tau$, we re-define B and τ as above and iterate. Each definition of B and τ requires $O(n)$ time by using median-finding techniques, whereas each computation of σ_j, $j \in B$, and j^* requires $O(|B|)$ time.

The solution S returned by GREEDY may contain redundant columns. This happens because the columns selected in a certain iteration to cover some uncovered rows, can lead a previously-selected column to become redundant. Removing the redundant columns in an optimal way leads to an SCP, defined by the redundant column set, say, R and by the set containing the rows covered only by the redundant columns. This problem can be solved either exactly through an enumerative algorithm for small values of $|R|$, or heuristically. We use the following mixed approach. When $|R| > 10$, we remove from S the column $j^* \in R$ with maximum cost c_{j^*}, update R, and repeat. As soon as $|R| \leq 10$, we use a simple enumeration scheme to remove, in an optimal way, the redundant columns left.

4 Column Fixing

Although typically very close to the optimum, the heuristic solution available at the end of the heuristic phase can often be improved by fixing in the solution a convenient set of columns, and re-applying the whole procedure to the resulting subproblem. To our knowledge, only the heuristic of Ceria, Nobili and Sassano (1995) makes use of variable fixing.

Name	Size	Best	Time	Col-Fix	No Col-Fix
RAIL582	$582 \times 55,515$	211	575.6*	211	213
RAIL507	$507 \times 63,009$	174	634.8*	176	180
RAIL2586	$2,586 \times 920,683$	947	486.1°	952	989
RAIL4872	$4,872 \times 968,672$	1534	854.4°	1550	1606

Table 1: Results of procedure 3-PHASE with and without column fixing – * Time in PC 486/33 CPU seconds – ° Time in HP 735/125 CPU seconds.

Clearly, the choice of the columns to be fixed is of crucial importance. After extensive computational testing, we decided to implement the following simple criteria. Let u^* be the best multiplier vector found during the subgradient and heuristic phases, and define $Q = \{j \in N : c_j(u^*) < -0.001\}$. We first fix each column $j \in Q$ for which there is a row i covered only by j among the columns in Q, i.e., $J_i \cap Q = \{j\}$. Then, we apply the heuristic procedure GREEDY(u^*, S) described in Section 3, and fix the first $\max\{\lfloor m/200 \rfloor, 1\}$ columns therein chosen.

Table 1 reports computational results for the 4 large scale instances from the FASTER test bed, described in more detail in Section 8.1. In the table, "Name" is the name of the instance, "Size" its size (in the format $m \times n$), and "Best" is the best solution value known. For each instance, we first ran procedure 3-PHASE yielding the solution value reported in column "Col-Fix", and then ran, for the same amount of time (column "Time"), the same procedure with no column fixing, yielding the solution value reported in column "No Col-Fix". The times in the table refer to the final version of our algorithm, which uses the pricing procedure described in the next section. The table clearly shows the effectiveness of column fixing, especially for the very large scale instances RAIL2586 and RAIL4872.

5 Pricing

In order to reduce the computing time spent by the overall algorithm, in all phases we work on a small subset of columns, defining the current core problem. This is of fundamental importance when large-scale instances are tackled, and gives substantial improvements even for medium-size instances. The approach is in the spirit of the well-known partial pricing technique for solving large-scale LP's. To our knowledge, however, this approach was never used in combination with Lagrangian relaxation for SCP. An example of the combined use of Lagrangian relaxation and column generation for crew scheduling can be found in Carraresi and Gallo (1986).

Our core problem is updated dynamically, by using the dual information associated

with the current Lagrangian multiplier vector u^k. To be specific, at the very beginning of the overall algorithm we define a "tentative" core by taking the first 5 columns covering each row. Afterwards, we work on the current core for, say, T consecutive subgradient iterations, after which we re-define the core problem as follows. We compute the Lagrangian cost $c_j(u^k)$, $j \in N$, associated with the current u^k, and define the column set of the new core as $C = C_1 \cup C_2$, where $C_1 = \{j \in N : c_j(u^k) < 0.1\}$, and C_2 contains the 5 smallest Lagrangian cost columns covering each row; if $|C_1| > 5m$, we keep in C_1 only the $5m$ smallest Lagrangian cost columns.

Notice that a valid lower bound for the overall problem is only available after the pricing step. The pricing phase requires $O(q)$ time, since the computation of the $5m$ lowest Lagrangian cost columns in C_1 can be done in $O(n)$ time by using median-finding techniques. As we work with a core problem containing $O(m)$ columns, the time complexity of each iteration of our subgradient procedure is $O(m \log m + p)$, while the time required by procedure GREEDY becomes $O(rm + p)$, where $p \leq q$ is the maximum number of nonzero entries of the SCP matrix for the core problem.

The pricing frequency is governed by parameter T. In our implementation we initially set $T = 10$. After each pricing, we compute $\Delta = (LB^k(C') - LB^k(N))/UB$, where UB is the best SCP solution value available, $LB^k(N) = \sum_{i \in M} u_i^k + \sum_{j \in N} \min\{c_j(u^k), 0\}$ is the "true" lower bound value available after pricing, and $LB^k(C') = \sum_{i \in M} u_i^k + \sum_{j \in C'} \min\{c_j(u^k), 0\}$ is its counterpart with respect to C', the column set of the core problem before pricing. If Δ is small, we decrease the pricing frequency by increasing T. If on the other hand Δ is large, i.e., the core problem is far from being settled, we reduce T. More precisely, we update $T = 10 \cdot T$ if $\Delta \leq 10^{-6}$; $T = 5 \cdot T$ if $\Delta \leq 0.02$; $T = 2 \cdot T$ if $\Delta \leq 0.2$; and $T = 10$ otherwise. In any case, the value T is not allowed to exceed $\min\{1000, m/3\}$.

According to our experience, for large scale instances the use of pricing cuts the overall computing time by more than one order of magnitude.

6 Refining Procedure

The solution available at the end of procedure 3-PHASE is typically close to the optimum, but in some cases it can be improved. For this purpose, we have defined a simple scheme for refining a given SCP solution. The scheme assigns a score to the chosen columns, fixes to 1 the variables associated with the best scored columns, and re-optimizes the resulting subproblem.

Let x^* define the best SCP solution computed by procedure 3-PHASE, and let u^* be the Lagrangian vector corresponding to the best lower bound computed for the overall

problem (that with no fixed column). The gap between the associated lower and upper bound values is given by

$$GAP = \sum_{j \in S} c_j - \left(\sum_{i \in M} u_i^* + \sum_{j \in N: c_j(u^*) < 0} c_j(u^*) \right)$$

$$= \sum_{j \in S} \left(c_j(u^*) + \sum_{i \in I_j} u_i^* \right) - \sum_{i \in M} u_i^* - \sum_{j \in N: c_j(u^*) < 0} c_j(u^*)$$

$$= \sum_{j \in S: c_j(u^*) > 0} c_j(u^*) \; + \; \sum_{j \in N \setminus S: c_j(u^*) < 0} |c_j(u^*)| \; + \; \sum_{i \in M} u_i^* \left(|S \cap J_i| - 1 \right),$$

where $S = \{j \in N : x_j^* = 1\}$. Accordingly, for each column j with $x_j^* = 1$ we compute an estimate δ_j of the contribution of j to the overall gap, namely

$$\delta_j = \max\{c_j(u^*), 0\} + \sum_{i \in I_j} u_i^* \frac{|S \cap J_i| - 1}{|S \cap J_i|}. \tag{9}$$

The second term in (9) is obtained by uniformly splitting, for each row i, the gap contribution $u_i^*(|S \cap J_i| - 1)$ among all the columns $l \in J_i$ with $x_l^* = 1$. Notice that this score cannot be computed "on line" in a greedy heuristic like procedure GREEDY.

In our view, columns with small δ_j are likely to be part of an optimal solution. Accordingly, we define the ordered column set $\{j \in N : x_j^* = 1\} = \{j_1, \ldots, j_p\}$, where $\delta_{j_1} \leq \delta_{j_2} \leq \ldots \leq \delta_{j_p}$, and find the first $j^* \in \{j_1, \ldots, j_p\}$ such that

$$\frac{|\bigcup_{j=j_1}^{j^*} I_j|}{m} \geq \pi, \tag{10}$$

where the parameter $\pi \in [0, 1]$ controls the percentage number of rows removed after fixing. We then fix columns $j_1, \ldots, j^*$, and re-optimize the resulting subproblem through procedure 3-PHASE.

7 The Overall Method

The above procedures are applied iteratively until a given time-limit is exceeded, or a sufficient precision is obtained, according to the following scheme:

Value	1550	1549	1547	1546	1544	1543	1542	1541	1540	1538	1537	1536	1535	1534
Time	733.7	866.1	866.3	866.6	868.6	881.6	882.4	937.8	1024.0	1150.0	1150.9	1187.8	4326.0	4556.1

Table 2: Incumbent solution value updating for instance RAIL4872 – Times in HP 735/125 CPU seconds – The first entry refers to the first application of procedure 3-PHASE.

Algorithm CFT;
begin

1. Initialize $z_{OPT} := +\infty$, $u^* := 0$, and $F := \emptyset$, where z_{OPT} is the value of the best available SCP solution, u^* is the best multiplier vector for the original SCP instance, and F is the set of columns fixed by the refining procedure;
   repeat
2. define the SCP subinstance $\mathcal{I}_F$ derived from fixing $x_j := 1$ for all $j \in F$;
3. apply procedure 3-PHASE$(\overline{x}, \overline{u})$ to $\mathcal{I}_F$;
4. **comment:** obtain the SCP solution x^* associated with the partial solution $\overline{x}$;
   for each $j \in N$ **do**
   if $j \in F$ **then** $x_j^* := 1$
   else $x_j^* := \overline{x}_j$;
5. **comment:** update the best SCP solution;
   if $\sum_{j \in N} c_j x_j^* < z_{OPT}$ **then begin**
   $z_{OPT} := \sum_{j \in N} c_j x_j^*$;
   $x_{OPT} := x^*$
   end
6. **comment:** define the new value for the threshold π;
   if $\sum_{j \in N} c_j x_j^* = z_{OPT}$ **or** $F = \emptyset$ **then** $\pi := \pi_{MIN}$
   else $\pi := \alpha \cdot \pi$;
7. **comment:** update the set F of columns fixed;
   for each $j \in \{k \in N : x_k^* = 1\}$ **do** compute δ_j as defined in (9);
   sort the set $\{j \in N : x_j^* = 1\}$ according to nondecreasing values δ_j and let $\{j_1, \ldots, j_p\}$ be the corresponding sorted set;
   define j^* as in (10), and let $F := \{j_1, \ldots, j^*\}$
   until $z_{OPT} \leq \beta \cdot L(u^*)$ **or** a time limit is exceeded

end.

A careful choice of parameters π_{MIN}, α and β is very important for the refining procedure to be effective. In our implementation, we set $\pi_{MIN} = 0.3$, $\alpha = 1.1$, and $\beta = 1.0$.

As an illustration of the effectiveness of the refining procedure, in Table 2 we report the updating of the incumbent solution value for instance RAIL4872, along with the associated computing times. The first entry refers to the first application of procedure 3-PHASE.

8 Computational Results

Algorithm *CFT* was implemented in ANSI FORTRAN 77. We store the whole SCP matrix columnwise, in sparse form, by using the vectors IA (of length $n+1$) and A (of length q), so as to have $I_j = \{A_{IA_j}, A_{IA_j+1}, \ldots, A_{IA_{j+1}-1}\}$ for $j = 1, \ldots, n$. We do not store the whole SCP matrix rowwise, while we store the matrix for the core problem both rowwise and columnwise, in sparse form. Unlike most existing algorithms for SCP, we do not perform any pre-processing on the initial data in order to remove dominated columns and rows. This is motivated by several reasons. First of all, pre-processing is very time consuming for large scale instances. Furthermore, thanks to the pricing procedure the running time of algorithm *CFT* does not change substantially if dominated rows and columns are also present. Finally, the score we use in procedure GREEDY and in the column-fixing phase prevents the choice of dominated columns.

8.1 Instances from the FASTER Competition

Algorithm *CFT* was first tested on the sample instances distributed by Ferrovie dello Stato SpA within the FASTER competition. Table 3 reports the corresponding results, where the entries have the following meaning:

Name is the name of the instance;

Size is the size of the instance in the format $m \times n$, where (μ) stands for micro, (m) for medium, and (l) for large;

Dens is the percentage number of entries equal to 1 in the SCP constraint matrix, i.e., q/mn (for the FASTER instances, $|I_j| \leq 12$ for each column j);

Range is the cost range in the format min-max;

LB is the lower bound on the optimal solution value computed by solving the LP relaxation of SCP, and rounding up the corresponding value;

Bologna reports the solution value found by algorithm *CFT* and the computing time spent up to the point where this solution is found for the first time;

					Bologna		Roma	Best Industry
Name	Size	Dens	Range	LB	Sol	Time	Sol	Sol
RAIL582	$582 \times 55,515$ (μ)	1.2%	1-2	210	211	570.0*	211	211
RAIL507	$507 \times 63,009$ (μ)	1.2%	1-2	173	175$^{\times}$	817.0*	174	174
RAIL2586	$2,586 \times 920,683$ (m)	0.4%	1-2	937	948^{+}	1183.2°	951	952
RAIL4872	$4,872 \times 968,672$ (l)	0.2%	1-2	1509	1534	4556.1°	1534	1538

Table 3: Results on FASTER sample instances – * Time in PC 486/33 CPU seconds – $^{\circ}$ Time in HP 735/125 CPU seconds – $^{\times}$ 174 with an ad hoc tuning – $^{+}$ 947 with an ad hoc tuning.

Roma reports the solution value found by the algorithm of the Dipartimento di Informatica e Sistemistica, University of Roma "La Sapienza";

Best Industry is the best solution value found by the SCP packages available on the market, as reported in Ceria, Nobili and Sassano (1995).

The group named Roma consisted of S. Ceria, P. Nobili and A. Sassano. They used a Lagrangian heuristic, based on the initial definition of a core problem which is not updated dynamically, and on a primal-dual subgradient technique combined with column fixing, see Ceria, Nobili and Sassano (1995). The computing time of Roma is within 1,000 CPU seconds on an IBM RS/6000 375 (i.e., about 15,000 seconds on a PC/486 33) for the micro instances, and within 10,000 CPU seconds (i.e., about 5,000 seconds on a HP 735/125) on the same machine for the medium and large instances (the authors do not report the computing time spent up to the point where the best solution is found for the first time).

The results for the competition are reported in Table 4, where the entries have the following meaning:

Name, **Size**, **Dens** and **Range** are as in Table 3;

LB is the rounded-up lower bound on the optimal solution value computed by algorithm *CFT*;

An reports the solution value found by the algorithm of the Istituto di Informatica, University of Ancona (A. Brunori, E. Faggioli and F. Pezzella);

Bo reports the solution value found by algorithm *CFT*;

Bo2 reports the solution value found by the algorithm of the Dipartimento di Matematica, University of Bologna (J. Beasley and A. Mingozzi);

Name	Size	Dens	Range	LB	An	Bo	Bo2	Fi	Rm	Tn
RAIL516 *	$516 \times 47,311$ (μ)	1.3%	1-2	182	188	182	$\times$	216	$\times^{+}$	$\times$
RAIL2536 °	$2,536 \times 1,081,841$ (m)	0.4%	1-2	685	745	691	765	922	692	709
RAIL4284 °	$4,284 \times 1,092,610$ (l)	0.2%	1-2	1051	1175	1065	1156	$\times$	1070	1117

Table 4: Results on FASTER competition instances – * Time limit: 3,000 seconds on a PC 486/33 – ° Time limit: 10,000 seconds on a HP 735/125 – $^{+}$ in Ceria, Nobili and Sassano, 1995, value 182 is reported.

Fi reports the solution value found by the algorithm of the Istituto di Matematica, University of Firenze (M. Antelmi Dazio, F. Carmusciano, A. Casavola and D. De Luca Cardillo);

Rm reports the solution value found by the algorithm of the Dipartimento di Informatica e Sistemistica, University of Roma "La Sapienza" (S. Ceria, P. Nobili and A. Sassano);

Tn reports the solution value found by the algorithm of the Istituto di Matematica, University of Trento (R. Battiti and G. Tecchiolli).

We report the results of all the groups whose code was successfully run by the Ferrovie dello Stato yielding feasible SCP solutions, according to the final report of the FASTER Jury. The entries with '$\times$' mean that the group did not partecipate in the competition for the given instance.

Algorithm *CFT* provided the best solution for all the instances. It is worth mentioning that for the medium and large instances, after a few hundred CPU seconds our solution was better than all the other final solutions, with the exception of those obtained by Roma. Moreover, when 1/6 of the time limit had elapsed, we had already obtained solution values, namely 182 for RAIL516, 691 for RAIL2536, and 1069 for RAIL4284, that were better than all the other final solution values. Our best solutions for instances RAIL2536 and RAIL4284 were found after about 1,200 and 8,000 CPU seconds, respectively.

8.2 Instances from the Literature

In order to analyze the effectiveness and robustness of algorithm *CFT*, we considered the SCP instances from the literature. We imposed a time limit of 5,000 CPU seconds on a DECstation 5000/240. As we will see, in most cases algorithm CFT is able to find an optimal solution, although it does not provide a proof of optimality.

					Be		LL		BaCa		BeCh		CFT	
Name	Size	Dens	Range	Opt	Sol	Time	Sol	Time	Sol	Time*	Sol	Time	Sol	Time
4.1	200 x 1,000	2%	1-100	429	429	11.0	429	0.5	429	0.8	429	294.8	429	2.3
4.2	200 x 1,000	2%	1-100	512	512	11.1	512	0.4	512	1.2	512	9.0	512	1.1
4.3	200 x 1,000	2%	1-100	516	516	6.8	516	0.3	516	1.0	516	16.4	516	2.1
4.4	200 x 1,000	2%	1-100	494	495	12.2	495	1.2	494	1.7	494	142.0	494	9.8
4.5	200 x 1,000	2%	1-100	512	512	7.0	512	0.3	512	0.5	512	44.1	512	2.1
4.6	200 x 1,000	2%	1-100	560	561	15.8	560	1.4	560	7.5	560	16.1	560	19.3
4.7	200 x 1,000	2%	1-100	430	430	9.2	430	0.5	430	1.0	430	138.6	430	2.7
4.8	200 x 1,000	2%	1-100	492	493	11.5	493	1.4	492	6.4	492	818.7	492	22.2
4.9	200 x 1,000	2%	1-100	641	641	20.6	641	1.5	641	9.2	641	136.1	641	1.8
4.10	200 x 1,000	2%	1-100	514	514	11.9	514	0.3	514	1.1	514	13.5	514	1.8
5.1	200 x 2,000	2%	1-100	253	255	17.4	253	1.5	254	11.1	253	42.1	253	3.3
5.2	200 x 2,000	2%	1-100	302	304	20.9	302	1.8	307	20.1	302	1332.6	302	2.3
5.3	200 x 2,000	2%	1-100	226	226	10.1	226	0.3	226	1.0	228	11.0	226	2.1
5.4	200 x 2,000	2%	1-100	242	242	11.5	242	1.5	243	11.4	242	10.1	242	1.9
5.5	200 x 2,000	2%	1-100	211	211	7.2	211	0.2	211	2.1	211	14.9	211	1.2
5.6	200 x 2,000	2%	1-100	213	213	11.3	213	0.4	213	1.3	213	29.9	213	0.9
5.7	200 x 2,000	2%	1-100	293	294	18.1	293	1.5	293	7.5	293	194.9	293	15.0
5.8	200 x 2,000	2%	1-100	288	288	20.7	288	1.6	288	4.3	288	3733.3	288	1.6
5.9	200 x 2,000	2%	1-100	279	279	15.7	279	0.8	279	1.5	279	13.5	279	2.6
5.10	200 x 2,000	2%	1-100	265	265	9.8	265	0.2	265	1.1	265	19.2	265	1.3
6.1	200 x 1,000	5%	1-100	138	141	16.8	138	1.7	140	9.2	138	46.1	138	22.6
6.2	200 x 1,000	5%	1-100	146	146	14.5	149	1.9	147	9.2	146	210.5	146	17.8
6.3	200 x 1,000	5%	1-100	145	145	15.0	145	1.6	145	11.5	145	11.8	145	2.3
6.4	200 x 1,000	5%	1-100	131	131	10.3	131	1.2	131	8.3	131	4.8	131	1.8
6.5	200 x 1,000	5%	1-100	161	162	13.3	161	2.1	163	10.4	161	12.1	161	2.2
A.1	300 x 3,000	2%	1-100	253	255	36.0	254	2.7	258	39.0	253	222.4	253	82.0
A.2	300 x 3,000	2%	1-100	252	256	44.2	255	2.9	254	40.9	252	327.9	252	116.2
A.3	300 x 3,000	2%	1-100	232	234	28.1	234	2.6	237	28.6	232	127.0	232	249.9
A.4	300 x 3,000	2%	1-100	234	235	33.5	234	2.4	235	36.3	234	45.5	234	4.7
A.5	300 x 3,000	2%	1-100	236	237	19.0	238	2.2	236	26.2	236	23.7	236	80.0
B.1	300 x 3,000	5%	1-100	69	70	28.4	70	3.0	69	29.0	69	20.0	69	4.0
B.2	300 x 3,000	5%	1-100	76	77	40.8	76	4.0	76	29.0	76	11.6	76	6.1
B.3	300 x 3,000	5%	1-100	80	80	25.4	81	4.4	81	35.1	80	709.7	80	18.0
B.4	300 x 3,000	5%	1-100	79	80	37.0	81	4.3	79	29.0	79	29.9	79	6.3
B.5	300 x 3,000	5%	1-100	72	72	26.0	72	4.1	72	32.6	72	5.3	72	3.3
C.1	400 x 4,000	2%	1-100	227	230	42.4	227	4.0	230	116.2	227	187.9	227	74.0
C.2	400 x 4,000	2%	1-100	219	223	66.0	222	4.1	220	56.1	219	40.7	219	64.2
C.3	400 x 4,000	2%	1-100	243	251	75.1	251	4.9	248	61.7	243	541.3	243	70.2
C.4	400 x 4,000	2%	1-100	219	224	63.4	224	5.4	224	68.1	219	144.6	219	61.6
C.5	400 x 4,000	2%	1-100	215	217	39.9	216	4.1	217	64.6	215	80.6	215	60.3
D.1	400 x 4,000	5%	1-100	60	61	40.9	60	4.8	61	36.6	60	13.8	60	23.1
D.2	400 x 4,000	5%	1-100	66	68	52.7	68	3.5	67	46.6	66	198.6	66	22.0
D.3	400 x 4,000	5%	1-100	72	75	55.8	75	5.8	74	47.2	72	785.3	72	22.6
D.4	400 x 4,000	5%	1-100	62	64	36.5	63	4.8	63	39.8	62	73.5	62	8.3
D.5	400 x 4,000	5%	1-100	61	62	36.7	62	4.5	61	36.2	61	79.8	61	10.3

Table 5: Results on the test instances from Beasley's OR-Library – Times are given in DECstation 5000/240 CPU seconds – * Overall time for the execution of the heuristic algorithm.

						Be		BaCa		CNS		BeCh		CFT	
Name	Size	Dens	Range	Best	LB	Sol	Time	Sol	Time*	Sol	Time°	Sol	Time	Sol	Time
E.1	500 x 5,000	10%	1-100	29	29	29	72.6	29	55.3	×	×	29	38.2	29	26.0
E.2	500 x 5,000	10%	1-100	30	28	32	92.7	32	76.0	×	×	30	14647.7	30	408.0
E.3	500 x 5,000	10%	1-100	27	27	28	92.7	28	80.9	×	×	27	28360.2	27	94.2
E.4	500 x 5,000	10%	1-100	28	28	30	100.3	29	77.5	×	×	28	539.9	28	26.3
E.5	500 x 5,000	10%	1-100	28	28	28	80.8	28	61.6	×	×	28	35.0	28	36.6
F.1	500 x 5,000	20%	1-100	14	14	15	43.9	14	67.5	×	×	14	76.4	14	33.2
F.2	500 x 5,000	20%	1-100	15	15	16	102.6	15	88.6	×	×	15	78.1	15	31.2
F.3	500 x 5,000	20%	1-100	14	14	15	124.7	15	76.5	×	×	14	266.8	14	248.5
F.4	500 x 5,000	20%	1-100	14	14	15	118.2	15	74.8	×	×	14	209.7	14	31.0
F.5	500 x 5,000	20%	1-100	13	13	14	129.3	14	62.2	×	×	13	13192.6	13	201.1
G.1	1,000 x 10,000	2%	1-100	176	165	184	287.8	183	325.6	176	4905.5	176	30200.0	176	147.0
G.2	1,000 x 10,000	2%	1-100	154	147	163	204.9	161	370.1	155	4905.5	155	360.5	154	783.4
G.3	1,000 x 10,000	2%	1-100	166	153	174	318.2	175	378.6	167	4905.5	166	7841.6	166	978.0
G.4	1,000 x 10,000	2%	1-100	168	154	176	292.0	176	332.2	170	4905.5	168	25304.7	168	378.5
G.5	1,000 x 10,000	2%	1-100	168	153	175	277.5	172	262.6	169	4905.5	168	549.3	168	237.2
H.1	1,000 x 10,000	5%	1-100	63	52	68	317.7	68	488.4	64	4905.5	64	1682.1	63	1451.1
H.2	1,000 x 10,000	5%	1-100	63	52	66	293.9	67	380.7	64	4905.5	64	530.3	63	887.0
H.3	1,000 x 10,000	5%	1-100	59	48	65	325.1	63	443.1	60	4905.5	59	1803.5	59	1560.3
H.4	1,000 x 10,000	5%	1-100	58	47	63	333.5	62	354.7	59	4905.5	58	27241.8	58	237.6
H.5	1,000 x 10,000	5%	1-100	55	46	60	303.0	58	321.3	55	4905.5	55	449.6	55	155.4

Table 6: Results on the test instances from Beasley's OR-Library – Times are given in DECstation 5000/240 CPU seconds – * Overall time for the execution of the heuristic algorithm – ° Time limit.

Tables 5 and 6 give the results for the instances from Beasley's OR Library (see Beasley, 1990). Since 1990, all the proposed algorithms for SCP have been mainly tested on these instances. We compare algorithm *CFT* with the genetic algorithm *BeCh* by Beasley and Chu (1996) yielding the best published solution values on all these instances. Comparison is also made with faster heuristics, namely algorithms *Be* by Beasley (1990), *BaCa* by Balas and Carrera (1996), and *CNS* by Ceria, Nobili and Sassano (1995), based on Lagrangian relaxation, and algorithm *LL* by Lorena and Lopes (1994) based on surrogate relaxation. Balas and Carrera give two different versions of their heuristic; we report the results of the version yielding better solution values, which on the other hand is typically three to five times slower.

For each algorithm, we report the computing time spent up to the point where its best solution is found for the first time, with the exception of algorithms *BaCa* and *CNS*. Balas and Carrera give the overall computing time spent on the execution of their heuristic procedure, while Ceria, Nobili and Sassano give the solution value obtained after a prefixed time limit. Furthermore, in Beasley and Chu (1996) the computational results refer to 10 trials for each instance, each corresponding to a different set of parameters for the genetic algorithm. For each instance, we report for algorithm *BeCh* an estimate of the time needed for finding the best solution, computed as follows. Among the 10 trials, we consider the first one, say the k-th, in which the best solution value is found,

and set the time to $(k-1) \cdot AET + AST$, where AET is the average execution time for a single trial, and AST is the average time for finding the best solution in a trial; both these average values are given in Beasley and Chu (1996).

Times in the tables are in DECstation 5000/240 CPU seconds, obtained by estimating the times of other machines according to the performances reported in Dongarra (1993). This leads to some unavoidable approximation when comparing codes running on different computers.

Table 5 reports the results for instances in Classes 4, 5, 6, A, B, C, and D, for which the optimal solution value is known (see column "Opt"). Ceria, Nobili and Sassano do not report any result for these instances. For all the instances, both algorithms *BeCh* and *CFT* find the optimum, with the exception of instance 5.3, for which *BeCh* finds a solution of value 228, while *CFT* finds an optimal solution of value 226. Algorithms *Be*, *LL* and *BaCa* give solution values which are on average slightly worse. The average computing time is about 2 seconds for algorithm *LL*, 25 seconds for *Be*, *BaCa* and *CFT*, and 250 seconds for *BeCh*. Therefore algoritm *CFT* is much faster than *BeCh*, while its speed is comparable to that of *Be* and *BaCa*. As to *LL*, it is by far the fastest algorithm, although the computing times reported in Lorena and Lopes (1994) do not include the time for the initial reduction. In 18 cases, however, this algorithm is not capable of finding the optimal solution.

Table 6 reports the results for larger instances (Classes E, F, G, and H) for which the optimal solution value was not known. In order to check the quality of the heuristic solutions, we tried to solve these instances by using the CPLEX 3.0 mixed integer optimizer on a DECstation 5000/240 with 32 Mbytes of core memory and 128 Mbytes of virtual memory. On each instance, we first ran CPLEX giving as initial upper bound the value of the best solution known. We set the time limit to 600,000 seconds, deactivated the node limit, and used the depth-first node selection. Column "Best" reports the value of the best solution found by CPLEX within the time limit (possibly, CPLEX did not improve the solution given on input). If optimality was proven, the same value is reported in column "LB". Otherwise, in order to improve the lower bound value, we ran again CPLEX with best-first node selection, reporting in column "LB" the best lower bound found in the two trials. Due to the difficulty of these instances, CPLEX always ran out of memory when best-first node selection was used. Timing is not reported for the CPLEX runs, which were made only for the purpose of producing an optimal solution or a tight lower bound. Anyway, the CPLEX runs took much longer than those of the heuristic algorithms. In particular, for the instances for which optimality was proven, the time required by CPLEX is about 1,000 times larger than that of algorithm CFT.

Lorena and Lopes do not report results for the instances in Table 6, while Ceria,

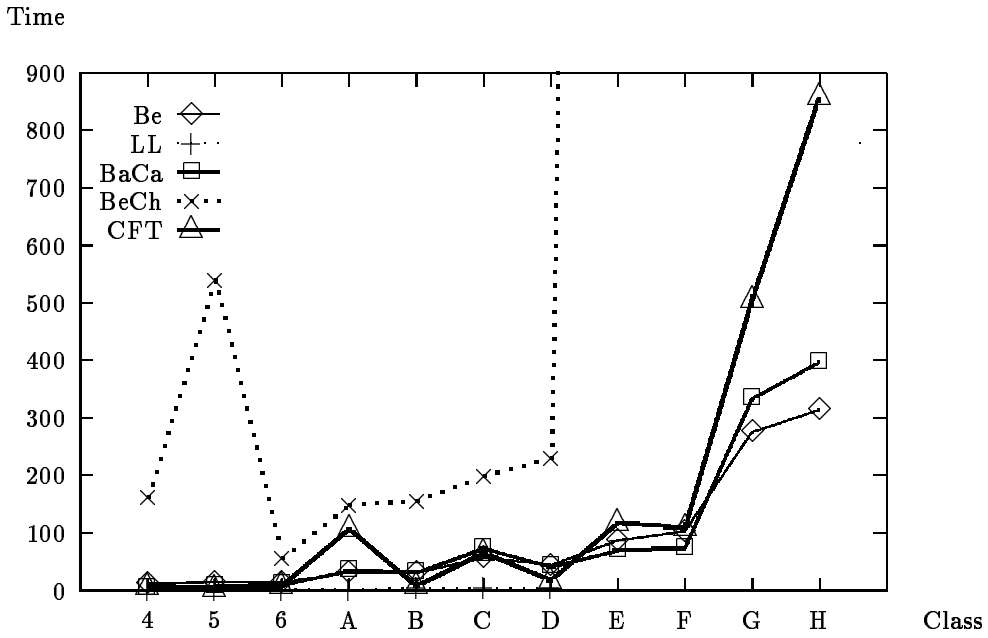

Figure 3: Comparison between the average computing times required by algorithms *Be*, *LL*, *BaCa*, *BeCh* and *CFT* on each instance class in Beasley's OR-Library – Times are given in DECstation 5000/240 CPU seconds.

Nobili and Sassano report results only for the instances in Classes G and H. For all these instances, algorithm *BeCh* yields the best solution known in the literature. Algorithm *CFT* always finds solutions that are at least as good as those of *BeCh*, yielding a better solution for instances G.2, H.1 and H.2. Algorithm *CNS* gives solutions which are on average slightly worse than those of *BeCh* and *CFT*, but better than those of *Be* and *BaCa*. The average computing time is about 200 seconds for algorithms *Be* and *BaCa*, 400 seconds for *CFT*, and 7500 seconds for *BeCh*. No fair comparison with the computing times of algorithm *CNS* can be made.

Figures 3 and 4 report an overall comparison of algorithms *Be*, *LL*, *BaCa*, *BeCh* and *CFT* on each instance class from Beasley's OR-Library. The lines between the points in these figures have no meaning, and are only given for the sake of readibility. The first figure gives the average computing time on each class, showing that algorithm *CFT* requires on average the same computing time as *Be* and *BaCa*, with the exception of Classes A, G and H. In this figure, we reported the computing time of *BeCh* up to Class D, since for the bigger classes the algorithm is at least one order of magnitude slower than the others. The second figure reports the average percentage gap between the solution found and the best one known, which is always found by algorithm *CFT*. This figure shows that, on these instances, the quality of the solutions found by algorithms *Be*, *LL* and *BaCa* considerably decreases when the size of the instances increases.

We also tested algorithm *CFT* on the SCP instances arising from crew scheduling

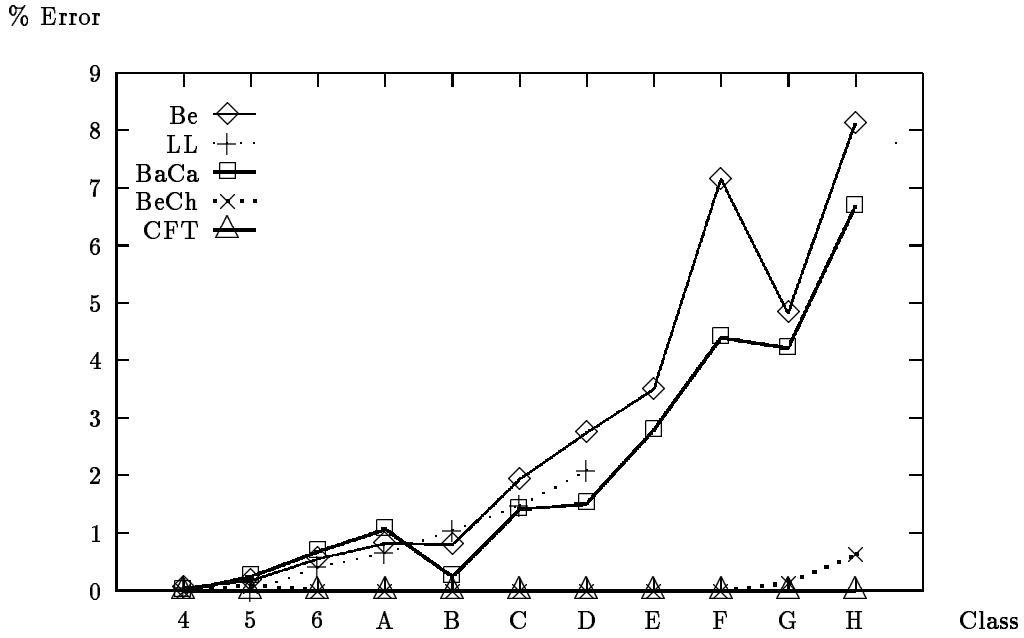

Figure 4: Comparison between the average percentage errors with respect to the best solution known for algorithms *Be*, *LL*, *BaCa*, *BeCh* and *CFT* on each instance class in Beasley's OR-Library.

in some airline companies, mentioned in Wedelin (1995), with the exception of instance SASD9imp1, for which we could not get the same data as reported in Wedelin (1995). In these instances, each column (pairing) has a cost taking into account several factors, and is expressed by very large numbers (with up to 7 digits). This is a substantial difference with respect to the previously-mentioned railway instances, where each column cost is either 1 or 2, and to the instances from the OR Library, where costs are always in $\{1,\ldots,100\}$. We compare algorithm *CFT* with algorithm *Wed* proposed by Wedelin (1995). For all instances but the last one, an optimal solution was found by CPLEX, and the corresponding values are reported in column "Opt" in Table 7. For instance SASD9imp2, this column reports the rounded-up solution value of the LP relaxation. Wedelin reports the computing time for the overall execution of his algorithm, but not that for finding the best solution for the first time; therefore a comparison of the computing times is not easy. The solution values are similar for all instances: for instances A320coc and SASD9imp2 *Wed* provides a slightly better solution, whereas the converse happened for instance SASjump. No computational result for Beasley's OR-Library test instances is reported in Wedelin (1995).

Finally, table 8 gives results for the real-world instances mentioned in Balas and Carrera (1996). These instances arise from crew scheduling applications, either in airline (Class 'AA') or bus (Class 'BUS') companies. For these instances, an optimal solution was found by Balas and Carrera's branch-and-bound algorithm, requiring, on average,

					Wed		CFT	
Name	Size	Dens	Range	Opt	Sol	Time°	Sol	Time
B727scratch	29 x 157	8.2%	1,600-11,850	94,400	94,400	4.7	94,400	0.3
ALITALIA	118 x 1,165	3.1%	2,200-2,110,900	27,258,300	27,258,300	37.2	27,258,300	6.2
A320	199 x 6,931	2.3%	1,600-2,111,450	12,620,100	12,620,100	216.9	12,620,100	79.5
A320coc	235 x 18,753	1.9%	1,900-1,812,000	14,495,500	14,495,500	1023.7	14,495,600	577.8
SASjump	742 x 10,370	0.6%	4,720-55,849	7,338,844	7,340,777	806.8	7,339,537	396.3
SASD9imp2	1,366 x 25,032	0.3%	3,860-35,200	5,261,088*	5,262,190	1579.7	5,263,640	2082.1

Table 7: Results on the airline instances from Wedelin – Times are given in DECstation 5000/240 CPU seconds – * LP lower bound – ° Overall time for the execution of the heuristic algorithm.

100 CPU seconds. We also report the results of Balas and Carrera's heuristic (again the version yielding better values). Algorithm *CFT* is capable of finding an optimal solution in all cases, within a computing time which is typically shorter (with a few exceptions) than that required for the overall execution of algorithm *BaCa*.

9 Conclusions

We have proposed a new heuristic for the set covering problem, based on Lagrangian relaxation. The algorithm uses subgradient optimization coupled with a very effective pricing technique for cutting computing-time. The convergence properties of the standard subgradient procedures are improved on by taking into account the massive dual degeneracy typically found in large scale instances from crew scheduling applications. In particular, we propose a new way of defining the ascent direction and the step size, allowing for a faster convergence to near-optimal dual solutions. We also describe and test computationally several score-definition rules to be used within greedy algorithms, exploiting the dual information available during subgradient optimization. A crucial step of our algorithm is that of variable fixing. Extensive computational testing shows the effectiveness of the fixing criteria we propose.

Although designed for solving very large scale instances coming from real-world railway applications, our algorithm proved very robust: in 92 out of 94 instances of our test bed we were able to find, within short computing time, the optimal (or the best known) solution. Moreover, among the 18 instances for which the optimum is not known, in 6 cases our solution is better than any other solution found by previous techniques. We do not know of any other commercial or academic code with comparable performance.

The proposed approach is also suitable for considering additional constraints arising in real-world applications. We are currently developing an extension of our method which takes into account several types of *base constraints* arising in railway crew scheduling

					BaCa		CFT	
Name	Size	Dens	Range	Opt	Sol	Time*	Sol	Time
AA03	$106 \times 8,661$	4.05%	91-3619	33,155	33,157	96.4	33,155	61.0
AA04	$106 \times 8,002$	4.05%	91-3619	34,573	34,573	39.2	34,573	3.6
AA05	$105 \times 7,435$	4.05%	91-3619	31,623	31,623	53.9	31,623	3.1
AA06	$105 \times 6,951$	4.11%	91-3619	37,464	37,464	44.4	37,464	5.2
AA11	$271 \times 4,413$	2.53%	35-2966	35,384	35,478	72.3	35,384	193.7
AA12	$272 \times 4,208$	2.52%	35-2966	30,809	30,815	48.0	30,809	53.8
AA13	$265 \times 4,025$	2.60%	35-2966	33,211	33,211	19.6	33,211	8.3
AA14	$266 \times 3,868$	2.50%	35-2966	33,219	33,222	86.2	33,219	30.3
AA15	$267 \times 3,701$	2.58%	35-2966	34,409	34,510	39.9	34,409	18.8
AA16	$265 \times 3,558$	2.63%	35-2966	32,752	32,858	54.5	32,752	33.6
AA17	$264 \times 3,425$	2.61%	35-2966	31,612	31,717	47.0	31,612	10.9
AA18	$271 \times 3,314$	2.55%	35-2966	36,782	36,866	66.2	36,782	13.5
AA19	$263 \times 3,202$	2.63%	35-2966	32,317	32,317	27.6	32,317	5.9
AA20	$269 \times 3,095$	2.58%	35-2966	34,912	35,160	34.4	34,912	13.6
BUS1	$454 \times 2,241$	1.89%	120-877	27,947	27,947	62.8	27,947	5.0
BUS2	$681 \times 9,524$	0.51%	120-576	67,760	67,868	356.0	67,760	19.2

Table 8: Results on the real-world instances from Balas and Carrera – Times are given in DECstation 5000/240 CPU seconds – * Overall time for the execution of the heuristic algorithm.

problems. Preliminary computational results are promising. More generally, our framework can be adapted to other 0-1 linear programs. Indeed, the main ideas of the scheme (Lagrangian optimization coupled with pricing, the use of Lagrangian dual information to drive a simple heuristic, variable fixing, and refining) can be applied in a more general context, provided that an effective heuristic algorithm is available as a substitute for procedure GREEDY.

Acknowledgements

We are grateful to Ferrovie dello Stato SpA, in particular to Pier Luigi Guida, for having organized the FASTER competition. We also thank Egon Balas and Anders Nilsson for providing us with the instances given in Tables 8 and 7, respectively, the anonymous referees for their useful comments, and Stefano Nucci for his help in programming.

References

[1] Associazione Italiana di Ricerca Operativa, Ferrovie dello Stato SpA, "Metodi di Ottimizzazione delle Risorse su Larga Scala - F.A.S.T.ER", Bando di Concorso, March 1994.

[2] Associazione Italiana di Ricerca Operativa, Ferrovie dello Stato SpA, "Verbale Commissione Concorso FASTER", July and September 1994.

[3] E. Balas, "A Class of Location, Distribution and Scheduling Problems: Modeling and Solution Methods", in P. Gray and L. Yuanzhang (eds.), *Proceedings of the Chinese-U.S. Symposium on Systems Analysis*, J. Wiley and Sons, 1983.

[4] E. Balas and M.C. Carrera, "A Dynamic Subgradient-Based Branch and Bound Procedure for Set Covering", *Operations Research* 44 (1996) 875–890.

[5] E. Balas and A. Ho, "Set Covering Algorithms Using Cutting Planes, Heuristics and Subgradient Optimization: A Computational Study", *Mathematical Programming Study* 12 (1980) 37–60.

[6] J.E. Beasley, "An Algorithm for Set Covering Problems", *European Journal of Operational Research* 31 (1987) 85–93.

[7] J.E. Beasley, "A Lagrangian Heuristic for Set Covering Problems", *Naval Research Logistics* 37 (1990) 151–164.

[8] J.E. Beasley, "OR-Library: Distributing Test Problems by Electronic Mail", *Journal of the Operational Research Society* 41 (1990) 1069–1072.

[9] J.E. Beasley and P.C. Chu, "A Genetic Algorithm for the Set Covering Problem", *European Journal of Operational Research* 94 (1996) 392–404.

[10] J.E. Beasley and K. Jörnsten, "Enhancing an Algorithm for Set Covering Problems", *European Journal of Operational Research* 58 (1992) 293–300.

[11] P. Carraresi and G. Gallo, "Optimization Models in Mass Transit Resource Management", *Ricerca Operativa* 38 (1986) 121–150.

[12] S. Ceria, P. Nobili, and A. Sassano, "A Lagrangian-Based Heuristic for Large-Scale Set Covering Problems", Technical Report R.406, 1995, IASI–CNR, Roma, to appear in *Mathematical Programming*.

[13] J.J. Dongarra, "Performance of Various Computers Using Standard Linear Equations Software", Technical Report No. CS-89-85, Computer Science Department, University of Tennessee, November 1993.

[14] M.L. Fisher, "The Lagrangian Relaxation Method for Solving Integer Programming Problems", *Management Science* 27 (1981) 1–18.

[15] M.L. Fisher and P. Kedia, "Optimal Solutions of Set Covering/Partitioning Problems Using Dual Heuristics", *Management Science* 36 (1990) 674–688.

[16] M. Held and R.M. Karp, "The Traveling Salesman Problem and Minimum Spanning Trees: Part II", *Mathematical Programming* 1 (1971) 6–25.

[17] L.A.N. Lorena and F.B. Lopes, "A Surrogate Heuristic for Set Covering Problems", *European Journal of Operational Research* 79 (1994) 138–150.

[18] P. Nobili and A. Sassano, "A Separation Routine for the Set Covering Polytope", in E. Balas, G. Cornuejols, and R. Kannan (eds.), *Integer Programming and Combinatorial Optimization*, Proceedings of the 2nd IPCO Conference, Carnegie-Mellon University Press, 1992.

[19] D. Wedelin, "An Algorithm for Large Scale 0-1 Integer Programming with Application to Airline Crew Scheduling", *Annals of Operational Research* 57 (1995) 283–301.