Annals of Operations Research 98, 353–371, 2000

Algorithms for the Set Covering Problem

ALBERTO CAPRARA and PAOLO TOTH
DEIS, University of Bologna, Viale Risorgimento 2, I-40136 Bologna, Italy

MATTEO FISCHETTI
DEI, University of Padova, Via Gradenigo 6/A, I-35131 Padova, Italy

Abstract. The *Set Covering Problem* (SCP) is a main model for several important applications, including crew scheduling in railway and mass-transit companies. In this survey, we focus our attention on the most recent and effective algorithms for SCP, considering both heuristic and exact approaches, outlining their main characteristics and presenting an experimental comparison on the test-bed instances of Beasley's OR Library.

1. Introduction

The *Set Covering Problem* (SCP) is a main model for several important applications. It is out of the scope of this survey to try to give a comprehensive list of all the practical problems which have been formulated and solved as SCP's: we refer the interested reader to the survey by Balas [1] and to the annotated bibliography by Ceria et al. [13]. Nevertheless, as of now, one of the most relevant applications of SCP is given by *crew scheduling* problems in railway and mass-transit transportation companies, 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, see, e.g., Caprara et al. [11].

In this survey, we focus our attention on the most recent and effective algorithms for SCP, considering both heuristic and exact approaches. For a comprehensive list of the algorithms proposed for the problem from the late 60's up to the late 80's, which are now out-to-date but have the merit of having inspired the current best approaches, we refer the reader to Ceria et al. [13]. The most effective algorithms presented in the literature (with few exceptions) have been computationally evaluated on the test-bed instances of Beasley's OR Library [5]. As the main aim of this work is to give an experimental comparison of existing SCP algorithms, a large part of it is devoted to illustrate the performance of these algorithms on the OR Library instances. Accordingly, we will not describe the methods which have not been tried on these instances.

SCP can formally be 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 natural mathematical model for SCP is

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

subject to

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

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

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

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

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

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

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, and note that generally q is much smaller than mn.

SCP is NP-hard in the strong sense, see Garey and Johnson [17], and also difficult from the point of view of theoretical approximation, see, e.g., Lund and Yannakakis [22]. Nevertheless, due to the structure of the real-world instances of the problem and to the considerable efforts spent to derive algorithms which perform better and better on these instances, the current state of the art on the problem is that instances with, say, a few hundred rows and a few thousand columns can be solved to proven optimality, and instances with, say, a few thousand rows and a few millions columns can be solved within about 1% of the optimum in a reasonable computing time.

The survey is organized as follows. Section 2 illustrates the linear programming and related relaxations of SCP, which are used by most of the effective heuristic and exact methods for the problem. We also give a comparison of a special-purpose approach to solve the linear programming relaxation of SCP with the state-of-the-art general-purpose linear programming solvers. In section 3 we describe the state-of-the-art heuristic approaches to the problem, comparing their results both considering the quality of the solutions and the running time. In section 4 we consider the best exact algorithms, all based on the branch-and-bound approach, comparing their execution times.

Many procedures are illustrated in the literature which reduce the size of an SCP instance by removing redundant columns and rows. The most commonly used procedures consider the removal of a column j such that there exists a column $k \neq j$ for which $I_j \subseteq I_k$ and $c_j \geqslant c_k$, the removal of a column j such that $|I_j| > 1$ and $c_j \geqslant \sum_{i \in I_j} \min\{c_k\colon k \in J_i\}$, the removal of a row i such that there exists a row $h \neq i$ for which $J_h \subseteq J_i$, and the inclusion in the solution of column j whenever $J_i = \{j\}$ for some

row i. These rules have to be applied in a careful way, as the straightforward implementation of the corresponding checks may be very time consuming for large scale instances, on the other hand they may considerably reduce the size of the instance at hand. Most of the algorithms we will present contain some preprocessing phase in which part of the above rules (or new ones) are used: since we would like to point out the essential structure of the various algorithms, we will not give the details of the preprocessing phase of each of them.

The OR Library contains 12 classes of SCP instances, namely classes 4, 5, 6, *e*, A, B, C, D, E, F, G and H. According to the results reported in the literature and to our computational experiments, the instances in classes 4, 5, 6 and *e* are quite easy, in the sense that in almost all cases the state-of-the-art heuristic algorithms find an optimal solution, and the state-of-the-art exact algorithms require a very short computing time. Therefore, we decided to give comparative results only for the 40 instances in classes A, B, C, D, E, F, G and H. In the tables, we report for each instance its name (column "Name"), its size in the format $m \times n$ ("Size"), its density, i.e., the percentage value of $q/(mn)$ ("Dens"), and the minimum and maximum value of an entry of cost vector c ("Range").

All sections present computational results on the OR Library instances, and in particular report running times expressed in DECstation 5000/240 CPU seconds. For the algorithms by the authors and the packages CPLEX and MINTO these are the actual running times measured in experiments carried out by the authors. For the algorithms by other authors, the times of other machines are converted into DECstation 5000/240 CPU seconds according to the performance reported in Dongarra [15]. This leads to some unavoidable approximation when comparing codes running on different computers. Anyway, even taking into account this very crude approximation, the running times reported give in most cases clear indications. In table 1 we give the performance reported in [15] for each machine used in some paper mentioned in this survey, along with the conversion factor ρ, which is multiplied by the running time reported by the authors to get the "equivalent" running time in DECstation 5000/240 CPU seconds.

2. Linear programming and other relaxations

As we will see, among the most effective heuristic and exact approaches to SCP, many are based on the solution of the *Linear Programming* (LP) relaxation of SCP, defined as (1) and (2) subject to

$$0 \leqslant x_j \leqslant 1, \quad j \in N. \tag{4}$$

Nevertheless, as the exact solution of this relaxation by general-purpose LP codes is typically rather time consuming, many SCP algorithms resort to Lagrangian relaxation

Table 1
Performance of various machines as reported in Dongarra [15] and conversion factor to DECstation 5000/240.

Machine	Reference	Mflop/s	ρ
DECstation 5000/240	[10]	5.3	1.00
Cray-1S	[3,4]	27	5.09
IBM PC-386	[21]	–	0.06*
DEC VAX 6000/410	[2]	1.2	0.22
SGI Indigo2	[6]	15	2.83
SUN Sparc10	[18]	10	1.89
IBM RISC/6000-375	[12]	26	4.90
HP 9000/835	[20]	1.8	0.34
PC Pentium 100	[8]	–	2.30*
Cray X-MP/28	[7]	106	20.00

* Performance evaluated through direct experiments.

combined with subgradient optimization in order to determine near-optimal solutions u of the dual of the LP relaxation, given by

$$\max\left\{\sum_{i\in M} u_i\colon \sum_{i\in I_j} u_i \leqslant c_j \ (j \in N),\ u_i \geqslant 0 \ (i \in M)\right\}. \tag{5}$$

The Lagrangian relaxation of model (1)–(3) is defined as follows. We assume the reader is familiar with Lagrangian relaxation theory (see, e.g., Fisher [16] for an introduction). For every vector $u \in R_+^m$ of Lagrangian multipliers associated with the constraints (2), the Lagrangian subproblem reads:

$$L(u) = \min \sum_{j\in N} c_j(u)x_j + \sum_{i\in M} u_i \tag{6}$$

subject to

$$x_j \in \{0, 1\}, \quad j \in N, \tag{7}$$

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 (6)–(7) 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 $c_j(u) = 0$. Note that, given u, $x(u)$ can be determined in $O(q)$ time. The Lagrangian dual problem associated with (6)–(7) consists of finding a Lagrangian multiplier vector $u^* \in R_+^m$ which maximizes the lower bound $L(u)$. As (6)–(7) has the *integrality property*, any optimal solution u^* of the dual of the LP relaxation of SCP is also an optimal solution of the Lagrangian dual problem (see Fisher [16]). As the only requirement of the Lagrangian multipliers is to be nonnegative, not all Lagrangian multiplier vectors correspond to feasible dual solutions. On the other hand, given any multiplier vector it is possible to compute a lower bound by taking into account the negative Lagrangian costs, which correspond to violated dual constraints. Moreover, given a multiplier vector, it is easy to modify it so as to obtain a feasible dual

solution without decreasing (and possibly by increasing) the associated lower bound. This can be done in a greedy way by selecting a column j such that $c_j(u) < 0$, suitably decreasing the value of u_i for $i \in I_j$, one after the other, until $c_j(u) = 0$, and repeating the procedure until no column having negative Lagrangian cost exists. For details, see, e.g., Balas and Carrera [2]. Therefore, the determination of near-optimal Lagrangian multipliers corresponds in some sense to a heuristic solution of the dual of the LP relaxation.

A well-known 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 multiplier vector u and the corresponding relaxed solution $x(u)$, defined by:

$$s_i(u) = 1 - \sum_{j \in J_i} x_j(u), \quad i \in M. \tag{8}$$

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, $k \geqslant 1$, a possible choice (Held and Karp [19]) consists of using the following simple updating formula:

$$u_i^{k+1} = \max\left\{u_i^k + \lambda \frac{UB - L(u^k)}{||s(u^k)||^2} s_i(u^k), 0\right\} \quad \text{for } i \in M, \tag{9}$$

where UB is an upper bound on v(SCP), and $\lambda > 0$ is a parameter that 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.

A commonly-used technique to reduce the size of an SCP instance is based on the observation that the Lagrangian cost $c_j(u)$ gives a lower bound on the increase of lower bound $L(u)$ if x_j is fixed to 1. Accordingly, one can fix x_j to 0 whenever $L(u) + c_j(u) \geqslant UB$. The same reasoning shows that one can fix x_j to 1 whenever $L(u) - c_j(u) \geqslant UB$ (recall that the Lagrangian cost may be negative). This technique is called *Lagrangian cost fixing*.

The above described standard subgradient optimization procedure can be improved in several ways, so as to achieve a faster convergence to a near-optimal multiplier vector. Below we illustrate a few variations presented in the literature.

Instead of relaxing the constraints (2) corresponding to all the rows in M, Balas and Carrera [2] keep explicitly in the relaxed problem the rows in a subset $\overline{M}$ such that $J_i \cap J_h = \emptyset$ for all $i, h \in \overline{M}$, $i \neq h$, and relax in a Lagrangian way the rows in $M \setminus \overline{M}$. It is easy to show that this relaxed problem can be solved in $O(q)$ time, and that the best lower bound obtainable through this relaxation is still equal to the LP relaxation value. Moreover, in [2] a so-called *dynamic* subgradient procedure is proposed, in which, at each iteration, the Lagrangian multiplier vector u is transformed into an LP dual feasible (and maximal) solution, as described above. By using Lagrangian cost fixing, the values of some variables are possibly fixed within the subgradient optimization procedure.

When very large instances are tackled, the computing time spent within subgradient optimization becomes very large. To overcome this difficulty, Caprara et al. [9] define

a *core problem* containing a suitable set of columns, chosen among those having the lowest Lagrangian costs, and use a *variable pricing* scheme to iteratively update the core problem in a vein similar to that used for solving large scale linear programs. The use of pricing within subgradient optimization drastically reduces computing time, and is one of the main ingredients for the success of the overall heuristic scheme proposed in [9], as discussed in the next section. Moreover, Caprara et al. [9] present improvements on the standard way of defining the step-size and the ascent direction within the subgradient optimization procedure.

A *primal-dual* subgradient approach is proposed by Ceria et al. [12]. In particular, one can restate the LP dual problem (5) by explicitly imposing the upper bound $\bar{c}_i = \max\{c_j\colon\ j \in J_i\}$ to each dual variable u_i, $i \in M$. If constraints $\sum_{i \in I_j} u_i \leqslant c_j$, $j \in N$, are relaxed in a Lagrangian way through a multiplier vector x, one obtains a Lagrangian dual problem which is equivalent to the primal LP relaxation of SCP, in the sense that an optimal primal solution yields an optimal multiplier vector. Accordingly, Ceria et al. [12] propose a subgradient optimization procedure which computes, at the same time, Lagrangian multiplier vectors for the original and the new Lagrangian problem, yielding both a lower bound and an upper bound on the LP value.

An alternative to Lagrangian relaxation is *surrogate relaxation*, which is used by Lorena and Lopes [21]. For a vector $v \in R_+^m$ of surrogate multipliers, the surrogate relaxed problem of SCP has the form:

$$S(u) = \min \sum_{j \in N} c_j x_j \tag{10}$$

subject to

$$\sum_{j \in N} \left(\sum_{i \in I_j} v_i \right) x_j \geqslant \sum_{i \in M} v_i, \tag{11}$$

$$x_j \in \{0, 1\}, \quad j \in N, \tag{12}$$

i.e., it is a *knapsack problem*, which is still NP-hard, but solvable in pseudo-polynomial time, and, in practice, quite effectively, see, e.g., Martello and Toth [23]. Nevertheless, Lorena and Lopes [21] further relax the integrality constraint (12) and solve, in $O(n)$ time, the continuous relaxation of the knapsack problem. Though the surrogate dual problem of the latter relaxation is still equivalent to the LP relaxation of SCP, the experimental results reported in Lorena and Lopes [21] suggest that the use of surrogate instead of Lagrangian relaxation within subgradient optimization allows one to obtain near-optimal multipliers in a shorter computing time.

Wedelin [26] presents a method alternative to subgradient optimization to compute near-optimal multipliers. This method considers one row i at the time, and updates the associated multiplier u_i as follows. For all $j \in J_i$ one computes the modified Lagrangian cost $c'_j(u) = c_j(u) + u_i$, takes the first and second smallest modified Lagrangian costs, say $c'_1(u)$ and $c'_2(u)$, and sets the new value of u_i to $(c'_1(u) + c'_2(u))/2$. The idea is to have exactly one column $j \in J_i$ with negative Lagrangian cost. By applying this operation

to each row (in turn) several times, one may get near-optimal multipliers. The actual method used by Wedelin [26] is a variation of this, in which two different values u_i^- and u_i^+ are used in place of u_i. The Lagrangian cost of the column $j \in J_i$ with negative Lagrangian cost after the last iteration on row i is computed by using u_i^-, whereas the Lagrangian costs of the remaining columns in J_i are computed by using u_i^+. This latter approach guarantees much better performance if embedded within a heuristic algorithm for SCP.

Most of the methods presented above are used within heuristic algorithms, i.e., their main objective is not to compute the tightest possible lower bound, but to drive the search of near-optimal SCP solutions. Therefore, it is not interesting to compare these methods according to the lower bound value they produce, rather we will compare the performance of the associated heuristic algorithms in the next section.

For several instances, the lower bound determined by Lagrangian or alternative relaxations is much worse than the optimal solution value of the LP relaxation. In our opinion, this is not a drawback within heuristic algorithms, where the main objective is to determine good SCP solutions and not to prove optimality. On the other hand, exact algorithms may behave much better if the LP relaxation is solved to optimality, as seen in section 4.

In Caprara et al. [10], a new approach to compute the exact solution of the LP relaxation of SCP is presented. The main idea is taken from the SPRINT approach described, e.g., in Chu et al. [14]. Here, one solves a sequence of LP relaxations associated with core problems, and uses a pricing procedure to test the possible optimality of the core problem solution for the whole problem, redefining the core problem if the solution is not optimal. The main novelties of the approach are the use of Lagrangian relaxation in order to initialize the core problem and the dual variables, and a multiplier adjusting technique which ensures a sequence of increasing lower bound values throughout the procedure.

In table 2 we report a computational comparison of the approach described above with the state-of-the-art LP solver CPLEX 4.0.8 on the instances of the OR Library. All the four LP algorithms provided by CPLEX have been tested, namely primal simplex (column “Primal”), dual simplex (“Dual”), network simplex (“Network”), and barrier method followed by crossover (“Barrier”). For each of these algorithms, we tuned the parameters so as to guarantee the best overall performance on the 40 instances of the OR library. We report the computational times of each of these algorithms in the corresponding column, as well as the time required by the approach of Caprara et al. [10] (column “CFT”) along with the corresponding time spent within internal calls to the CPLEX LP solver (“Simplex”). The average computing time is about 420 seconds for “Primal”, 370 seconds for “Barrier”, 46 seconds for “Dual” and “Network”, and 10 seconds for “CFT” (with about 7 seconds for the CPLEX calls). Therefore, on average, the new approach is faster than the best CPLEX algorithms by a factor of almost 5.

Table 2
LP algorithms.

Name	Size	Dens (%)	Range	Value	CPLEX				CFT	
					Primal	Dual	Network	Barrier	Total	Simplex
A.1	300 × 3000	2.0	1–100	246.8	5.16	2.85	2.76	8.90	1.42	0.80
A.2	300 × 3000	2.0	1–100	247.5	6.05	4.09	3.74	8.45	1.87	1.07
A.3	300 × 3000	2.0	1–100	228.0	5.25	3.92	3.38	8.54	1.42	0.80
A.4	300 × 3000	2.0	1–100	231.4	6.59	3.92	3.83	8.63	1.42	0.80
A.5	300 × 3000	2.0	1–100	234.9	5.34	3.56	3.47	8.45	1.33	0.71
B.1	300 × 3000	5.0	1–100	64.5	14.24	6.23	6.23	15.04	1.60	0.80
B.2	300 × 3000	5.0	1–100	69.3	12.37	7.03	6.67	15.57	1.87	1.07
B.3	300 × 3000	5.0	1–100	74.2	14.77	6.23	6.05	15.22	1.51	0.71
B.4	300 × 3000	5.0	1–100	71.2	11.57	6.23	6.14	14.68	1.51	0.80
B.5	300 × 3000	5.0	1–100	67.7	12.90	6.05	5.96	15.66	1.25	0.71
C.1	400 × 4000	2.0	1–100	223.8	12.28	5.61	5.34	18.69	2.14	1.25
C.2	400 × 4000	2.0	1–100	212.8	11.48	5.78	5.70	19.49	2.14	1.25
C.3	400 × 4000	2.0	1–100	234.6	13.97	7.03	6.41	23.14	2.40	1.42
C.4	400 × 4000	2.0	1–100	213.8	11.21	6.05	5.87	18.78	2.76	1.78
C.5	400 × 4000	2.0	1–100	211.6	11.39	6.85	6.41	22.07	2.14	1.25
D.1	400 × 4000	5.0	1–100	55.3	27.32	11.12	11.39	32.48	2.85	1.60
D.2	400 × 4000	5.0	1–100	59.3	32.31	10.95	10.77	33.55	2.85	1.60
D.3	400 × 4000	5.0	1–100	65.1	37.82	11.04	11.04	34.17	3.29	2.05
D.4	400 × 4000	5.0	1–100	55.8	29.55	10.50	10.41	36.76	3.11	1.87
D.5	400 × 4000	5.0	1–100	58.6	27.23	10.95	10.59	35.24	2.31	1.16
E.1	500 × 5000	10.0	1–100	21.4	298.85	37.91	37.02	181.46	5.96	2.76
E.2	500 × 5000	10.0	1–100	22.4	241.27	39.34	40.23	236.82	7.30	4.09
E.3	500 × 5000	10.0	1–100	20.5	220.89	39.07	38.09	201.40	7.48	3.65
E.4	500 × 5000	10.0	1–100	21.4	288.26	39.78	39.43	205.23	6.23	2.85
E.5	500 × 5000	10.0	1–100	21.3	300.72	38.36	38.18	204.34	6.76	3.56
F.1	500 × 5000	20.0	1–100	8.8	702.45	87.66	91.76	1100.71	11.57	2.49
F.2	500 × 5000	20.0	1–100	10.0	746.32	87.75	86.15	1403.03	12.46	2.85
F.3	500 × 5000	20.0	1–100	9.5	760.56	85.26	92.29	1097.32	12.19	3.47
F.4	500 × 5000	20.0	1–100	8.5	721.94	85.70	91.40	1073.92	11.84	2.85
F.5	500 × 5000	20.0	1–100	7.8	588.27	87.66	90.95	986.35	12.19	3.29
G.1	1000 × 10000	2.0	1–100	159.9	676.37	70.93	66.75	519.56	24.65	21.54
G.2	1000 × 10000	2.0	1–100	142.1	448.54	63.37	60.43	499.09	20.83	16.02
G.3	1000 × 10000	2.0	1–100	148.3	420.06	61.41	63.72	526.15	25.72	20.74
G.4	1000 × 10000	2.0	1–100	148.9	526.95	65.06	60.78	609.18	20.02	16.82
G.5	1000 × 10000	2.0	1–100	148.2	540.83	66.30	63.28	527.75	22.34	19.13
H.1	1000 × 10000	5.0	1–100	48.1	1696.54	144.89	147.29	952.17	33.91	23.67
H.2	1000 × 10000	5.0	1–100	48.6	1750.65	147.64	160.99	1031.56	36.40	26.34
H.3	1000 × 10000	5.0	1–100	45.2	1884.67	150.14	158.32	1025.50	33.91	24.21
H.4	1000 × 10000	5.0	1–100	44.0	1885.92	150.85	137.14	945.23	30.08	20.65
H.5	1000 × 10000	5.0	1–100	42.4	1743.17	139.19	141.68	1075.25	27.05	17.27

3. Heuristic algorithms

In this section, we illustrate the most effective heuristic algorithms for SCP which have been experimentally evaluated on the test problems of the OR Library. Many of these heuristics are based on the following observation. For a near-optimal Lagrangian multiplier vector u, the Lagrangian cost $c_j(u)$ gives reliable information on the overall utility of selecting column j. Based on this property, the Lagrangian (rather than the original) costs are used 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, for a given Lagrangian multiplier vector u, 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.

The greedy approach which is common to all the algorithms below is the following. A solution S is initialized to be empty, and the set M' of uncovered rows is set equal to M. Then, iteratively, the column j with the best value of a score σ_j is added to S, and M' is updated. Score σ_j is typically a function of the original cost c_j, of the number of rows in M' covered by column j, and of the multipliers associated with these rows. At the end of this procedure, set S typically contains a set R of *redundant* columns, i.e., columns j such that $S \setminus \{j\}$ is still a feasible SCP solution. The optimal removal of the redundant columns amounts to solving an SCP defined by the columns in R and the rows in $M \setminus \bigcup_{j \in S \setminus R} I_j$.

The algorithm proposed by Beasley [4] computes, at every iteration of a subgradient optimization procedure, a feasible SCP solution as follows. A set S is initialized with all the columns selected in the solution of the Lagrangian problem. Then, for each row i not covered by S, the column with smallest original cost in J_i is added to S. Finally, the columns in S are considered in decreasing order of original costs, and a column j is removed from S if $S \setminus \{j\}$ is a feasible SCP solution. At each iteration, Lagrangian cost fixing is performed in the attempt of removing some columns from the problem.

The algorithm by Balas and Carrera [2] is based on the dynamic subgradient procedure described in the previous section. The version producing the best heuristic solutions applies, at every iteration, a greedy heuristic similar to the one used by Beasley [4]. Recall from the previous section that in [2] not all the constraints are relaxed in a Lagrangian way, and that the subgradient procedure works with multiplier vectors which are feasible dual solutions. Accordingly, the solution of the relaxed problem contains all the columns with zero Lagrangian costs and the columns with smallest cost covering the rows which are not relaxed. This solution, say S, is completed by either adding the columns with smallest Lagrangian cost covering each row uncovered by S, or by adding columns in decreasing order of c_j/k_j values, where k_j is the number of uncovered rows

covered by column j. Finally, redundant columns are removed in an order which is not specified in the paper.

The heuristic algorithm of Lorena and Lopes [21] uses the surrogate relaxation described in the previous section, and embeds within the subgradient optimization procedure a greedy heuristic identical to the one proposed by Beasley [4].

The approach of Ceria et al. [12] is mainly intended for large scale SCP instances. Initially, a subgradient optimization procedure is applied to the whole problem so as to define a good core problem. Differently from Caprara et al. [9], where the initial selection of the core problem is not a critical issue since this is updated dynamically, Ceria et al. [12] determine the core problem at the beginning in a careful way and never change it afterwards. The heuristic algorithm performed on the core problem applies primal-dual subgradient optimization, fixes to 1 a variable chosen accordingly to its Lagrangian cost and its value in the primal multiplier vector, finds in a greedy way a feasible solution starting from the one containing the variables fixed to 1, and iterates. When the columns corresponding to the variables fixed to 1 yield a feasible solution, the process is restarted, without redefining the core problem, by changing some parameters for the variable selection and the greedy heuristic parts.

The heuristic approach proposed by Haddadi [18] is a variation of the one by Beasley [4]. In particular, at each iteration of the subgradient optimization procedure, the solution of the Lagrangian problem is made feasible by a greedy heuristic based on the original costs. Then, the redundant columns are removed from the solution by solving the associated SCP: it is not specified in the paper if this (typically small) SCP is solved to proven optimality, or a heuristic procedure is used.

The approach by Caprara et al. [9] consists of three main phases. 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, as outlined in the previous section. The second one is the *heuristic phase*, in which a sequence of near-optimal Lagrangian vectors is determined and, for each vector, the associated Lagrangian costs are given on input to a greedy heuristic procedure. In the third phase, called *column fixing*, one fixes to 1 the value of the variables associated with the first k columns selected by the greedy heuristic. In this way one obtains an SCP instance with a reduced number of columns and rows, on which the three-phase procedure is iterated. The above three phases are iterated until the solution obtained cannot be improved further. After each application of the three phases, a *refining procedure* is used, which in some cases produces improved solutions. Throughout the algorithm, one works with a core problem defined by a small column subset, which is periodically re-defined as described in the previous section.

Even if most of the successful heuristic approaches to SCP are based on Lagrangian relaxation, there are some relevant exceptions which are worth mentioning.

Beasly and Chu [6] follow a genetic approach for the solution of the problem. Their algorithm maintains a *population* of SCP solutions encoded as 0-1 vectors x. Initially, a random population of p solutions is generated. Then, at every iteration, two solutions x^1 and x^2 are randomly selected from the current population and combined in a

third solution x^3 such that $x^3_j = x^1_j$ whenever $x^1_j = x^2_j$, and $x^3_j = x^1_j$ with probability $c(x^2)/(c(x^1) + c(x^2))$ if $x^1_j \neq x^2_j$, where $c(x)$ denotes the cost corresponding to x. After the definition of x^3 from x^1 and x^2, a certain number of randomly-selected components of x^3 are changed, and then x^3 is transformed into a feasible minimal solution in a greedy way, analogous to the one used by Beasley [4]. Finally, the new solution x^3 is added to the current population in place of a randomly-chosen previous solution. The process is stopped after a certain number of iterations. An alternative genetic algorithm has been proposed by Al-Sultan et al. [25].

Both approaches used by Jacobs and Brusco [20] and Brusco et al. [8] are based on simulated annealing. In the first paper [20], an initial solution S is generated by a greedy algorithm which, at every iteration, randomly selects an uncovered row and adds to the solution the column with smallest index covering that row. After this addition, possible redundant columns in the solution are removed, and the process is iterated until a feasible (minimal) solution is determined. Then, a number of iterations are performed in which d randomly-chosen columns are removed from the current solution S, which is then completed so as to obtain a new (minimal) solution S' in a greedy way, again analogous to the one used by Beasley [4]. S' becomes the current solution if it is better than S, otherwise S is replaced by S' with a probability which decreases exponentially with the difference between the values of S and S', and with the number of iterations performed. The two main enhancements of Brusco et al. [8] over the approach of Jacobs and Brusco [20] are the following. First of all, the selection of the columns to be removed from the current solution S is randomly performed (as before) every third iteration. In the remaining two iterations one removes the columns whose removal leaves the smallest number of uncovered rows. Second, for each column j in the solution one has a list of so-called *morphs*, which are columns "similar" to j. After the columns are removed from the current solution, and after the addition of every fourth column to complete the partial solution, each column in the solution is replaced by one of its morphs if this improves the ratio between the cost of the partial solution and the number of rows covered in the partial solution.

Table 3 reports the results for instances in classes A, B, C, and D of the OR Library, for which the optimal solution value is known (see column "Opt"). In particular, for the algorithms of Beasley [4] (column "Be"), Lorena and Lopes [21] ("LL"), Balas and Carrera [2] ("BaCa"), Beasley and Chu [6] ("BeCh"), Haddadi [18] ("Ha") and Caprara et al. [9] ("CFT"), we report the value of the best solution found (column "Sol"), as well as the time elapsed when the best solution was found ("Time"), with the exception of algorithms "BaCa" and "Ha", for which the time for the execution of the whole heuristic is given. Ceria et al. [12], Jacobs and Brusco [20] and Brusco et al. [8] report no result for these instances. For all the instances, both algorithms "BeCh" and "CFT" find the optimum. Algorithms "Be", "LL", "BaCa" and "Ha" give solution values which are on average slightly worse. The average computing time is about 4 seconds for algorithm "LL", 30 seconds for "Ha", 40 seconds for "Be", 45 seconds for "BaCa", 50 seconds for "CFT", and 180 seconds for "BeCh". Therefore

Table 3
Heuristic algorithms for classes A, B, C, D.

Name	Size	Dens (%)	Range	Opt	Be		LL		BaCa		BeCh		Ha		CFT	
					Sol	Time	Sol	Time	Sol	Time*	Sol	Time	Sol	Time*	Sol	Time
A.1	300 × 3, 000	2	1–100	253	255	36.0	254	2.7	258	39.0	253	222.4	254	22.2	253	82.0
A.2	300 × 3, 000	2	1–100	252	256	44.2	255	2.9	254	40.9	252	328.0	253	20.3	252	116.2
A.3	300 × 3, 000	2	1–100	232	234	28.1	234	2.6	237	28.6	232	127.1	234	21.9	232	249.9
A.4	300 × 3, 000	2	1–100	234	235	33.5	234	2.4	235	36.3	234	45.6	234	16.8	234	4.7
A.5	300 × 3, 000	2	1–100	236	237	19.0	238	2.2	236	26.2	236	23.8	236	17.4	236	80.0
B.1	300 × 3, 000	5	1–100	69	70	28.4	70	3.0	69	29.0	69	20.1	69	26.4	69	4.0
B.2	300 × 3, 000	5	1–100	76	77	40.8	76	4.0	76	29.0	76	11.6	76	28.4	76	6.1
B.3	300 × 3, 000	5	1–100	80	80	25.5	81	3.6	81	35.1	80	709.8	81	28.0	80	18.0
B.4	300 × 3, 000	5	1–100	79	80	37.0	81	4.5	79	29.0	79	30.0	79	29.7	79	6.3
B.5	300 × 3, 000	5	1–100	72	72	26.0	72	2.9	72	32.6	72	5.4	72	26.7	72	3.3
C.1	400 × 4, 000	2	1–100	227	230	42.4	227	4.0	230	116.2	227	187.9	228	32.8	227	74.0
C.2	400 × 4, 000	2	1–100	219	223	66.0	222	4.1	220	56.1	219	40.8	223	34.0	219	64.2
C.3	400 × 4, 000	2	1–100	243	251	75.1	251	4.9	248	61.7	243	541.4	245	35.9	243	70.2
C.4	400 × 4, 000	2	1–100	219	224	63.4	224	5.4	224	68.1	219	144.6	223	32.8	219	61.6
C.5	400 × 4, 000	2	1–100	215	217	39.9	216	4.1	217	64.6	215	80.7	216	30.2	215	60.3
D.1	400 × 4, 000	5	1–100	60	61	40.9	60	4.8	61	36.6	60	13.9	61	46.0	60	23.1
D.2	400 × 4, 000	5	1–100	66	68	52.7	68	6.7	67	46.6	66	198.7	66	49.4	66	22.0
D.3	400 × 4, 000	5	1–100	72	75	55.8	75	5.8	74	47.2	72	785.3	73	48.6	72	22.6
D.4	400 × 4, 000	5	1–100	62	64	36.5	63	4.8	63	39.8	62	73.6	62	48.0	62	8.3
D.5	400 × 4, 000	5	1–100	61	62	36.7	62	4.4	61	36.2	61	79.8	62	42.6	61	10.3

* Overall time for the execution of the heuristic algorithm.

Table 4
Heuristic algorithms for classes E, F, G, H.

Name	Size	Dens (%)	Range	Best	LB	Be		BaCa		CNS		BeCh		JaBr		Ha		BJT		CFT	
						Sol	Time	Sol	Time*	Sol	Time°	Sol	Time	Sol	Time°	Sol	Time*	Sol	Time	Sol	Time
E.1	500 × 5, 000	10	1–100	29	29	29	72.6	29	55.3	×	×	29	38.2	29	408.0	29	187.2	29	0.8	29	26.0
E.2	500 × 5, 000	10	1–100	30	28	32	92.7	32	76.0	×	×	30	14647.8	30	408.0	32	199.9	30	23.7	30	408.0
E.3	500 × 5, 000	10	1–100	27	27	28	92.7	28	80.9	×	×	27	28360.3	27	408.0	28	198.2	27	31.1	27	94.2
E.4	500 × 5, 000	10	1–100	28	28	30	100.3	29	77.5	×	×	28	540.0	28	408.0	29	191.3	28	4.0	28	26.3
E.5	500 × 5, 000	10	1–100	28	28	28	80.8	28	61.6	×	×	28	35.1	28	408.0	28	181.7	28	1.4	28	36.6
F.1	500 × 5, 000	20	1–100	14	14	15	141.3	14	67.5	×	×	14	76.4	14	408.0	14	654.1	14	2.4	14	33.2
F.2	500 × 5, 000	20	1–100	15	15	16	102.6	15	88.6	×	×	15	78.1	15	408.0	15	646.8	15	1.9	15	31.2
F.3	500 × 5, 000	20	1–100	14	14	15	124.7	15	76.5	×	×	14	266.9	14	408.0	15	665.4	14	7.2	14	248.5
F.4	500 × 5, 000	20	1–100	14	14	15	118.2	15	74.8	×	×	14	209.7	14	408.0	15	665.5	14	4.0	14	31.0
F.5	500 × 5, 000	20	1–100	13	13	14	129.3	14	62.2	×	×	13	13192.6	14	408.0	14	684.9	13	180.7	13	201.1
G.1	1, 000 × 10, 000	2	1–100	176	165	184	287.8	183	325.6	176	4900.0	176	30200.1	179	408.0	181	191.2	176	135.3	176	147.0
G.2	1, 000 × 10, 000	2	1–100	154	147	163	204.9	161	370.1	155	4900.0	155	360.5	158	408.0	160	177.9	155	112.7	154	783.4
G.3	1, 000 × 10, 000	2	1–100	166	153	174	318.2	175	378.6	167	4900.0	166	7841.6	170	408.0	174	193.1	166	1755.1	166	978.0
G.4	1, 000 × 10, 000	2	1–100	168	154	176	292.1	176	332.2	170	4900.0	168	25304.7	172	408.0	177	184.0	168	1792.3	168	378.5
G.5	1, 000 × 10, 000	2	1–100	168	153	175	277.5	172	262.6	169	4900.0	168	549.3	168	408.0	173	192.0	168	160.7	168	237.2
H.1	1, 000 × 10, 000	5	1–100	63	52	68	317.7	68	488.5	64	4900.0	64	1682.2	64	408.0	66	479.9	64	44.6	63	1451.1
H.2	1, 000 × 10, 000	5	1–100	63	52	66	293.9	67	380.7	64	4900.0	64	530.3	64	408.0	67	462.3	63	643.7	63	887.0
H.3	1, 000 × 10, 000	5	1–100	59	48	65	325.1	63	443.1	60	4900.0	59	1803.6	61	408.0	62	462.4	59	255.4	59	1560.3
H.4	1, 000 × 10, 000	5	1–100	58	47	63	333.5	62	354.7	59	4900.0	58	27241.9	59	408.0	61	458.0	58	139.2	58	237.6
H.5	1, 000 × 10, 000	5	1–100	55	46	60	303.0	58	321.4	55	4900.0	55	449.7	55	408.0	56	452.3	55	23.4	55	155.4

* Overall time for the execution of the heuristic algorithm.
° Time limit.

algoritm “CFT” is much faster than “BeCh”, while its speed is comparable to that of “Be”, “BaCa” and “Ha”. As to “LL”, it is by far the fastest algorithm, although the computing times reported in Lorena and Lopes [21] do not include the time for the initial reduction. In 15 of the 20 cases, however, this algorithm is not capable of finding the optimal solution.

Table 4 reports the results for the larger instances of the OR Library (classes E, F, G, and H) for which the optimal solution value is known only in a few cases. In columns “Best” and “LB” we report the best known solution value and the best known lower bound value, respectively. With respect to table 3, the entry for the algorithm of Lorena and Lopes [21] is missing, as they do not report results for these instances, whereas we have new entries for the algorithms of Jacobs and Brusco [20] (column “JaBr”), Brusco et al. [8] (“BJT”) and Ceria et al. [12] (“CNS”). The latter authors report results only for the instances in classes G and H. Moreover, Jacobs and Brusco [20] and Ceria et al. [12] give only the time limit for the execution of their algorithm, which we report in column “Time”. For all these instances, algorithm “CFT” yields the best solution known in the literature. Algorithms “BeCh” and “BJT” find the same solutions as “CFT” with three and two exceptions, respectively. Algorithms “CNS” and “JaBr” give solutions which are on average slightly worse than those of “BeCh”, “BJT” and “CFT”, but better than those of “Be”, “BaCa” and “Ha”. The average computing time is about 200 seconds for algorithms “Be” and “BaCa”, 250 seconds for “BJT”, 400 seconds for “Ha” and “CFT”, and 7500 seconds for “BeCh”. No fair comparison with the computing times of algorithm “CNS” and “JaBr” can be made.

4. Exact algorithms

The most effective exact approaches to SCP are branch-and-bound algorithms in which lower bounds are computed by solving the LP relaxation of SCP, possibly in a dual heuristic way, as explained in section 2. In particular, all the algorithms which have been tested on the instances from the OR Library are of this type. The main reason for the success of these approaches is the fact that, despite the LP lower bound is not always very strong for these instances, it is apparently very difficult to get significantly stronger lower bounds by alternative methods which are computationally more expensive. This also explains the competitiveness of general-purpose ILP packages such as CPLEX or MINTO, which are based on branch-and-bound with the solution of an LP relaxation at each node of the branching tree, and include a lot of implementation effort in order to speed-up the parametric re-optimization of the LP’s solved after the root node.

We next describe the state-of-the-art exact algorithms for SCP in some detail. Before doing this, we would like to cite the algorithm of Nobili and Sassano [24], which is the only one, to our knowledge, which tries to obtain stronger lower bounds than the LP relaxation value by using special-purpose cutting planes, exploiting the structure of the SCP polytope. Unfortunately, although the algorithm was tested on the same instances considered by the approaches mentioned below, the authors report only the number of

nodes and not the computing time required by their method, therefore no fair comparison with the other approaches can be made.

The algorithm proposed by Beasley [3] solves the LP relaxation of SCP to optimality at the root node of the branching tree, whereas at the other nodes a lower bound is computed by using Lagrangian relaxation along with subgradient optimization, starting with the multiplier vector associated with the parent node. At the root node, in particular, near-optimal Lagrangian multipliers are computed through a dual ascent procedure followed by a subgradient procedure. Then the problem is reduced in size by applying Lagrangian cost fixing, and the LP relaxation of the reduced problem is solved to optimality by the dual simplex method. Apparently, the solution of the LP relaxation takes no advantage of the information obtained from the computation of near-optimal Lagrangian multipliers. Namely, the only reason for computing these multipliers is the smaller LP which has to be solved after the reduction. Feasible solutions are computed by using simple greedy heuristics, and several dominance procedures to reduce both the number of rows and columns are applied. The branching rule selects the uncovered row with the highest Lagrangian multiplier, and fixes to 0 and to 1, respectively, the variable associated with the column having the smallest Lagrangian cost covering the selected row.

An enhancement of the algorithm of Beasley [3] is given in Beasley and Jörnsten [7]. A main difference with respect to the original algorithm concerns the addition of constraints, not necessarily of the same type as (2), to the problem. Two of these constraints are added to exclude the best feasible solution found at the end of the subgradient optimization procedure at the root node. They impose that not all the columns in this solution can be selected, and that at least one column not in this solution must be selected. After the addition of these constraints, the LP relaxation is solved as in [3], and then classical Gomory fractional cuts, associated with the fractional LP solution, are added. These constraints are handled by the subgradient optimization procedure applied at the nodes other than the root. Other enhancements with respect to [3] concern the use of a the Lagrangian heuristic of [4] and a new branching strategy. The latter computes at each node the Lagrangian solution corresponding to the best multiplier vector u, determines the row i with maximum $|s_i(u)u_i|$, where $s_i(u)$ is the subgradient associated with row i, as defined in (8), and branches by fixing to 0 and to 1, respectively, the variable associated with the column $j \in J_i$ such that $x_j(u) = 1$ and the value of x_j is maximum in the LP solution at the root node (after the addition of the Gomory cuts).

The most recent exact algorithm for SCP presented in the literature is the one by Balas and Carrera [2]. This algorithm is based on the embedding of the dynamic subgradient procedure described in section 2 into a branch-and-bound scheme, and on the application of the subgradient optimization procedure at every node of the branching tree. The authors use two binary branching schemes based on the dual information associated with the Lagrangian multipliers and aimed at having a large lower bound increase at both descendant nodes. Notice that the best Lagrangian multiplier vector u computed

by the method of Balas and Carrera [2] always corresponds to a feasible (and maximal) dual solution, i.e., there are no columns with negative Lagrangian cost. The first branching scheme defines the set N^0 of the columns with zero Lagrangian cost and the set $J \subseteq N^0$ of the columns j such that there exists at least one row i with $u_i > 0$ for which $J_i \cap N^0 = \{j\}$. If $J \neq \emptyset$, one branches by fixing to 0 and to 1, respectively, the variable associated with the column $j \in J$ which maximizes $\max\{|J_i \cap N^0|\colon i \in I_j\}$. The main idea is that if x_j is fixed to 0, then the Lagrangian multipliers associated with the rows i such that $J_i \cap N^0 = \{j\}$ can be increased. On the other hand, if x_j is fixed to 1, the rows in I_j can be removed, and the reduced costs of other columns covering these rows are increased, giving the possibility of increasing other Lagrangian multipliers keeping all the reduced costs nonnegative. If $J = \emptyset$, the second branching scheme is applied. This scheme determines the row i such that $J_i \cap N^0 \neq J_i$ with minimum $|J_i \cap N^0|$. Note that such a row i may not exist, i.e., all the columns may have zero reduced cost, but this typically does not happen in practice. On the first branch one fixes to 0 all the variables associated with the columns in $J_i \cap N^0$, whereas on the second branch one replaces row i by a new row i' with $J_{i'} = J_i \cap N^0$, i.e., imposes that row i is covered only by columns in $J_i \cap N^0$. On the first branch, again the Lagrangian multiplier of row i can be increased, whereas on the second branch one gets a tighter constraint than before, that hopefully allows for a lower bound increase when subgradient optimization is applied.

We next present a comparison of the methods described above on the instances of classes A, B, C and D of the OR Library, which are those currently solvable to proven optimality in a reasonable computing time. In table 5 we report the computing time (column "Time") and number of branch-and-bound nodes explored ("Nodes") by the algorithms by Beasley [3] ("Be"), Beasley and Jörnsten [7] ("BeJö") and Balas and Carrera [2] ("BaCa"). We also consider the general-purpose ILP solvers CPLEX 4.0.8 and MINTO 2.3, which are run by giving them on input both the original instances (see columns "CPLEX" and "MINTO"), and the instances preprocessed by applying the heuristic algorithm of Caprara et al. [9] (without post-optimization), the method of Caprara et al. [10] for the computation of the LP relaxation (see section 2), and an LP reduced-cost fixing. This procedure provides on input to the ILP solvers an instance of considerably smaller size than the original one, along with a near-optimal SCP solution and the optimal LP solution (see columns "Impr. CPLEX" and "Impr. MINTO"). According to the table, the average computing time is about 3000 seconds for algorithm "Be", 2000 seconds for "BeJö", 1500 seconds for "BaCa", 700 seconds for "CPLEX", 750 seconds for "MINTO", 500 seconds for "Impr. MINTO", and 400 seconds for "Impr. CPLEX". This shows that the state-of-the-art general-purpose ILP solvers are competitive with the best exact algorithms for SCP presented in the literature, and that their performance can sensibly be improved by an external preprocessing procedure.

Table 5
Exact algorithms for classes A, B, C, D.

Name	Size	Dens	Range	Opt	Be		BeJö		BaCa		CPLEX		MINTO		Impr. CPLEX		Impr. MINTO	
					Time	Nodes	Time	Nodes	Time	Nodes	Time	Nodes	Time	Nodes	Time	Nodes	Time	Nodes
A.1	300 × 3, 000	2	1–100	253	395.0	515	565.2	373	175.5	95	137.6	44	66.3	55	63.4	29	44.1	55
A.2	300 × 3, 000	2	1–100	252	533.9	809	555.6	357	48.0	28	56.1	18	78.9	85	53.8	27	55.4	97
A.3	300 × 3, 000	2	1–100	232	564.0	857	482.4	377	66.6	41	91.3	37	66.5	71	45.6	25	44.8	71
A.4	300 × 3, 000	2	1–100	234	158.3	63	126.6	17	18.3	10	26.7	10	25.0	13	8.4	6	11.7	13
A.5	300 × 3, 000	2	1–100	236	144.6	79	94.8	15	7.7	3	22.4	10	24.6	13	13.7	14	14.1	13
B.1	300 × 3, 000	5	1–100	69	417.9	411	413.2	137	105.3	81	165.1	40	141.1	67	72.0	32	69.1	67
B.2	300 × 3, 000	5	1–100	76	2019.7	3273	1578.6	1655	1264.7	939	759.9	173	470.1	301	631.5	206	255.4	301
B.3	300 × 3, 000	5	1–100	80	761.5	1383	774.8	671	347.5	288	252.7	66	232.2	261	153.4	69	152.4	261
B.4	300 × 3, 000	5	1–100	79	2870.8	4381	2505.4	3157	2076.5	1508	1245.1	301	1419.6	1403	741.3	298	1034.8	1403
B.5	300 × 3, 000	5	1–100	72	460.6	661	459.8	355	94.6	69	220.4	56	119.7	93	112.2	57	74.4	93
C.1	400 × 4, 000	2	1–100	227	1035.3	1265	1038.4	549	86.7	34	169.4	38	90.0	39	108.6	46	45.4	45
C.2	400 × 4, 000	2	1–100	219	575.7	275	1242.0	619	356.5	145	308.6	61	204.1	91	246.0	71	119.2	92
C.3	400 × 4, 000	2	1–100	243	10396.3	15757	5950.8	5957	667.8	291	683.5	129	599.3	489	238.2	73	512.3	485
C.4	400 × 4, 000	2	1–100	219	661.7	493	1063.0	515	296.8	116	181.6	38	71.7	27	151.2	47	38.5	27
C.5	400 × 4, 000	2	1–100	215	547.7	375	1233.4	1001	201.7	91	63.5	13	111.8	47	119.9	34	52.9	47
D.1	400 × 4, 000	5	1–100	60	2987.8	3393	1351.8	1015	599.8	337	1484.8	210	965.4	324	326.3	84	404.0	305
D.2	400 × 4, 000	5	1–100	66	5376.1	6739	6718.4	7179	5323.1	2771	3638.7	575	4046.3	2791	1858.9	496	3084.1	2791
D.3	400 × 4, 000	5	1–100	72	15124.4	19707	9439.0	10455	11102.7	5120	2179.1	324	3117.6	2055	1504.5	358	2188.7	2055
D.4	400 × 4, 000	5	1–100	62	9914.8	12389	7535.6	8687	6218.5	2931	2252.4	315	3148.9	1957	1131.0	272	2078.2	1955
D.5	400 × 4, 000	5	1–100	61	387.3	147	454.4	63	37.6	18	336.9	43	79.8	13	106.4	28	31.4	13

References

[1] E. Balas, A class of location, distribution and scheduling problems: Modeling and solution methods, in: *Proceedings of the Chinese–U.S. Symposium on Systems Analysis*, eds. P. Gray and L. Yuanzhang (Wiley, 1983).

[2] E. Balas and M.C. Carrera, A dynamic subgradient-based branch-and-bound procedure for set covering, Operations Research 44 (1996) 875–890.

[3] J.E. Beasley, An algorithm for set covering problems, European Journal of Operational Research 31 (1987) 85–93.

[4] J.E. Beasley, A Lagrangian heuristic for set covering problems, Naval Research Logistics 37 (1990) 151–164.

[5] J.E. Beasley, OR-Library: Distributing test problems by electronic mail, Journal of the Operational Research Society 41 (1990) 1069–1072.

[6] J.E. Beasley and P.C. Chu, A genetic algorithm for the set covering problem, European Journal of Operational Research 94 (1996) 392–404.

[7] J.E. Beasley and K. Jörnsten, Enhancing an algorithm for set covering problems, European Journal of Operational Research 58 (1992) 293–300.

[8] M.J. Brusco, L.W. Jacobs and G.M. Thompson, A morphing procedure to supplement a simulated annealing heuristic for cost- and coverage-correlated weighted set-covering problems, Annals of Operations Research 86 (1999) 611–627.

[9] A. Caprara, M. Fischetti and P. Toth, A heuristic method for the set covering problem, Operations Research 47 (1999) 730–743.

[10] A. Caprara, M. Fischetti and P. Toth, Effective solution of the LP relaxation of set covering problems, Working Paper, DEIS, University of Bologna (1998).

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

[12] S. Ceria, P. Nobili and A. Sassano, A Lagrangian-based heuristic for large-scale set covering problems, Mathematical Programming 81 (1998) 215–228.

[13] S. Ceria, P. Nobili and A. Sassano, Set covering problem, in: *Annotated Bibliographies in Combinatorial Optimization*, eds. M. Dell'Amico, F. Maffioli and S. Martello (Wiley, 1997) pp. 415–428.

[14] H.D. Chu, E. Gelman and E.L. Johnson, Solving large scale crew scheduling problems, European Journal of Operational Research 97 (1997) 260–268.

[15] J.J. Dongarra, Performance of various computers using standard linear equations software, Technical Report No. CS-89-85, Computer Science Department, University of Tennessee (1996).

[16] M.L. Fisher, An applications oriented guide to Lagrangian optimization, Interfaces 15 (1985) 10–21.

[17] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness* (Freeman, 1979).

[18] S. Haddadi, Simple Lagrangian heuristic for the set covering problem, European Journal of Operational Research 97 (1997) 200–204.

[19] M. Held and R.M. Karp, The traveling salesman problem and minimum spanning trees: Part II, Mathematical Programming 1 (1971) 6–25.

[20] L.W. Jacobs and M.J. Brusco, A local search heuristic for large set-covering problems, Naval Research Logistics 52 (1995) 1129–1140.

[21] L.A.N. Lorena and F.B. Lopes, A surrogate heuristic for set covering problems, European Journal of Operational Research 79 (1994) 138–150.

[22] C. Lund and M. Yannakakis, On the hardness of approximating minimization problems, Journal of the ACM 41 (1994) 960–981.

[23] S. Martello and P. Toth, *Knapsack Problems: Algorithms and Computer Implementations* (Wiley, 1990).

[24] P. Nobili and A. Sassano, A separation routine for the set covering polytope, in: *Integer Programming and Combinatorial Optimization*, eds. E. Balas, G. Cornuejols and R. Kannan, Proceedings of the 2nd IPCO Conference (Carnegie-Mellon University Press, 1992).
[25] K.S. Al-Sultan, M.F. Hussain and I.S. Nizami, A genetic algorithm for the set covering problem, Journal of the Operational Research Society 47 (1996) 702–709.
[26] D. Wedelin, An algorithm for large scale 0-1 integer programming with application to airline crew scheduling, Annals of Operations Research 57 (1995) 283–301.