Journal of the Operations Research
Society of Japan
2007, Vol. 50, No. 4, 350-375

RELAXATION HEURISTICS FOR THE SET COVERING PROBLEM

Shunji Umetani
University of Electro-Communications

Mutsunori Yagiura
Nagoya University

(Received December 19, 2006; Revised June 15, 2007)

Abstract The set covering problem (SCP) is one of representative combinatorial optimization problems, which has many practical applications. The continuous development of mathematical programming has derived a number of impressive heuristic algorithms as well as exact branch-and-bound algorithms, which can solve huge SCP instances of bus, railway and airline crew scheduling problems. We survey heuristic algorithms for SCP focusing mainly on contributions of mathematical programming techniques to heuristics, and illustrate their performance through experimental analysis.

Keywords: Combinatorial optimization, set covering problem, linear programming, Lagrangian relaxation, subgradient method, heuristic algorithm

1. Introduction

The *set covering problem* (SCP) is one of representative combinatorial optimization problems, which has many practical applications, e.g., bus, railway and airline crew scheduling, vehicle routing, facility location, political districting [5, 23, 32]. More recent applications of SCP are found in probe selection in hybridization experiments in DNA sequencing [16] and feature selection and pattern construction in logical analysis of data [17].

SCP is formally defined as follows. We are given a ground set of m elements $i \in M = \{1, \dots, m\}$, a collection of n subsets $S_j \subseteq M$ and costs c_j for $j \in N = \{1, \dots, n\}$, where we assume $c_j > 0$ and $|S_j| \geq 1$ without loss of generality. We say that $X \subseteq N$ is a cover of M if $\bigcup_{j \in X} S_j = M$ holds, and X is a prime cover of M if X is a cover of M without any redundant subset. The goal of SCP is to find a minimum cost cover X of M. SCP can be formulated as a 0-1 integer programming (IP) problem as follows:

$$\begin{array}{lll} \text{(SCP)} & \text{minimize} & z(\boldsymbol{x}) = \sum_{j \in N} c_j x_j \\ & \text{subject to} & \sum_{j \in N} a_{ij} x_j \geq 1 \qquad (i \in M) \\ & & x_j \in \{0, 1\} \qquad (j \in N), \end{array} \tag{1.1}$$

where $a_{ij} = 1$ if $i \in S_j$ and $a_{ij} = 0$ otherwise; i.e., a column $\boldsymbol{a}_j = (a_{1j}, \dots, a_{mj})^{\mathrm{T}}$ of matrix (a_{ij}) represents the corresponding subset S_j by $S_j = \{i \in M \mid a_{ij} = 1\}$. It is understood that decision variable $x_j = 1$ if a subset S_j is selected in the cover X and $x_j = 0$ otherwise. For notational convenience, for each $i \in M$, let $N_i = \{j \in N \mid a_{ij} = 1\}$ be the set of subsets that contain element i. Since a column $j \in N$ and a row $i \in M$ correspond to a subset S_j and an element i respectively, we say that a column j covers a row i if $a_{ij} = 1$ holds.

SCP is known to be NP-hard in the strong sense [31], and there is no polynomial time approximation scheme (PTAS) unless P = NP. Furthermore, a number of lower bounds on approximation ratio for SCP have been shown. Feige [27] proved that, for any $\epsilon > 0$, it is impossible to achieve a polynomial time $(1-\epsilon)\ln n$ approximation algorithm unless NP has $n^{O(\log\log n)}$-time deterministic algorithms, and Trevisan [46] showed that the problem is hard to approximate within a factor $\ln d - O(\ln\ln d)$ unless P = NP, where $d = \max_{j\in N}|S_j|$. However, theoretical results do not necessarily reflect the experimental performance in practice; e.g., we can improve the performance of algorithms for the real-world instances by utilizing their structural property. The continuous development of mathematical programming has much improved the performance of exact branch-and-bound algorithms [3, 4, 7, 12, 30] accompanying with advances in computational machinery. Recent exact branch-and-bound algorithms enable us to solve large SCP instances with up to 400 rows and 4000 columns exactly [3].

Heuristic algorithms have also been studied extensively [2, 8, 11, 18, 28], and several efficient metaheuristic algorithms have been developed to solve huge SCP instances with up to 5000 rows and 1,000,000 columns within about 1% of the optimum in a reasonable computing time [20, 22, 24, 50]. Most of these impressive results were achieved by hybridizing metaheuristics and mathematical programming approaches. For example, Beasley [8] presented a number of greedy algorithms based on a Lagrangian relaxation (called the Lagrangian heuristics), and Caprara et al. [20] introduced variable fixing and pricing techniques into a Lagrangian heuristics.

In this paper, we present a review of heuristic algorithms and related mathematical programming techniques for SCP. We mainly focus on contributions of mathematical programming to heuristic algorithms for SCP, and illustrate their performance through experimental evaluation for several classes of benchmark instances.

We tested these algorithms on an IBM-compatible personal computer (Intel Xeon 2.8GHz, 2GB memory), and used two well-known sets of benchmark instances. The first set of benchmark instances is Beasley's OR Library [9], which contains 11 classes of SCP instances, namely 4–6 and A–H. Each of classes 4 and 5 has 10 instances, and each of classes 6 and A–H has 5 instances. In this paper, we denote instances in class 4 as 4.1, 4.2, ..., 4.10, and other instances in classes 5, 6 and A–H similarly. Another set of benchmark instances is called RAIL arising from a crew pairing problem in an Italian railway company [20, 24], which contains seven instances, namely three small size instances with up to 600 rows and 60,000 columns (RAIL507, 516 and 582), two medium size instances with up to 2600 rows and 1,100,000 columns (RAIL2536 and 2586), and two large size instances with up to 4900 rows and 1,100,000 columns (RAIL4284 and 4872). The data of these instances are given in Table 1, where density is defined by $\sum_{i\in M}\sum_{j\in N} a_{ij}/mn$.

This paper is organized as follows. In Section 2, we review representative relaxation techniques for SCP called the linear programming (LP) relaxation, the Lagrangian relaxation and the surrogate relaxation. In Section 3, we explain a well-known approach called the subgradient method, which computes good lower bounds of SCP instances within a short computing time. We also illustrate several techniques for improving the performance of the subgradient method. In Section 4, we explain a pricing method called the sifting method to solve huge relaxation problems of SCP. In Section 5, we illustrate problem reduction rules that test feasibility and reduce the size of SCP instances. In Section 6, we review heuristic algorithms for SCP including construction algorithms, Lagrangian heuristics and the state-of-the-art algorithms based on hybridization of metaheuristics and mathematical programming approaches.

Table 1: The size, density and cost range of benchmark instances for SCP

Instance	Rows	Columns	Density(%)	Cost range
4.1–4.10	200	1000	2	[1,100]
5.1–5.10	200	2000	2	[1,100]
6.1–6.5	200	1000	5	[1,100]
A.1–A.5	300	3000	2	[1,100]
B.1–B.5	300	3000	5	[1,100]
C.1–C.5	400	4000	2	[1,100]
D.1–D.5	400	4000	5	[1,100]
E.1–E.5	500	5000	10	[1,100]
F.1–F.5	500	5000	20	[1,100]
G.1–G.5	1000	10,000	2	[1,100]
H.1–H.5	1000	10,000	5	[1,100]
RAIL507	507	63,009	1.3	[1,2]
RAIL516	516	47,311	1.3	[1,2]
RAIL582	582	55,515	1.2	[1,2]
RAIL2536	2536	1,081,841	0.4	[1,2]
RAIL2586	2586	920,683	0.3	[1,2]
RAIL4284	4284	1,092,610	0.2	[1,2]
RAIL4872	4872	968,672	0.2	[1,2]

2. Relaxation Problems

The relaxation problems give us useful information to solve SCP. We can directly obtain good lower bounds from their solutions, and also good upper bounds by modifying them. In this section, we review three well-known relaxation problems for SCP called the linear programming (LP) relaxation problem, the Lagrangian relaxation problem, and the surrogate relaxation problem.

2.1. Linear programming relaxation

The most general technique is the *linear programming* (LP) relaxation, which is defined by replacing the binary constraints $x_j \in \{0,1\}$ with $0 \le x_j \le 1$ for all $j \in N$. Since the upper bound on x_j is known to be redundant, we can obtain the following LP relaxation problem:

$$\begin{array}{lll} \text{(LP)} & \text{minimize} & z_{\text{LP}}(\boldsymbol{x}) = \sum_{j \in N} c_j x_j \\ & \text{subject to} & \sum_{j \in N} a_{ij} x_j \ge 1 \qquad (i \in M) \\ & & x_j \ge 0 \qquad (j \in N). \end{array} \tag{2.1}$$

Although general-purpose LP solvers give an exact solution of the LP relaxation problem, it has been pointed out in the literature that their computation would be quite expensive because these solvers often suffer various problems caused by the degeneracy of the LP relaxation of SCP instances. However, in recent years, the development of mathematical programming softwares, accompanied with advances in computing machinery, enables us to solve huge LP instances [14]. We accordingly report a computational comparison of the simplex and barrier (or interior-point) methods with two state-of-the-art general purpose LP solvers called GLPK 4.8 (GNU Linear Programming Kit) [52] and CPLEX 9.1.3 [51] on the benchmark instances.

Table 2 illustrates the optimal value z_{LP} of the LP relaxation problem and computation time in seconds spent by GLPK and CPLEX with the modes of the primal and dual simplex methods and the barrier method, where we set the time limit to be 3600 seconds for each

Table 2: Comparison of the simplex and barrier methods

Instance	Rows	Columns	z_{LP}	GLPK 4.8			CPLEX 9.1.3		
				Primal	Dual	Barrier	Primal	Dual	Barrier
4.1–4.10	200	1000	509.10	0.20	0.01	0.15	0.02	0.01	0.02
5.1–5.10	200	2000	256.38	0.40	0.01	0.25	0.02	0.01	0.02
6.1–6.5	200	1000	139.22	0.35	0.02	0.37	0.07	0.01	0.04
A.1–A.5	300	3000	237.73	1.50	0.04	1.08	0.13	0.03	0.08
B.1–B.5	300	3000	69.38	2.62	0.06	2.70	0.36	0.05	0.13
C.1–C.5	400	4000	219.34	3.81	0.08	3.00	0.44	0.05	0.18
D.1–D.5	400	4000	58.84	8.06	0.11	6.78	1.11	0.06	0.26
E.1–E.5	500	5000	21.38	28.92	0.24	28.71	9.10	0.19	0.99
F.1–F.5	500	5000	8.92	47.12	0.31	57.60	23.74	0.40	5.41
G.1–G.5	1000	10,000	149.48	276.54	1.20	79.89	27.95	0.47	3.16
H.1–H.5	1000	10,000	45.67	653.31	1.38	145.47	90.65	0.81	4.80
RAIL507	507	63,009	172.15	53.00	14.19	15.31	6.78	10.48	1.99
RAIL516	516	47,311	182.00	19.09	2.81	13.75	2.55	4.02	1.44
RAIL582	582	55,515	209.71	59.09	6.23	25.83	7.61	8.31	2.16
RAIL2536	2536	1,081,841	688.40	>3600	>3600	>3600	2393.34	1673.13	146.06
RAIL2586	2586	920,683	935.92	>3600	>3600	>3600	1450.06	1734.22	105.98
RAIL4284	4284	1,092,610	1054.05	>3600	>3600	>3600	>3600	3517.55	298.70
RAIL4872	4872	968,672	1509.64	>3600	>3600	>3600	>3600	3546.61	155.44

Note: each result of classes 4–6 and A–H reports the average for all instances in the class.

run. In Table 2, computation time does not include the time for reading instance data, and each result of classes 4–6 and A–H reports the average computation time for all instances in the class, while the results for individual instances are reported for class RAIL. This is the same for all computational results in this paper.

We observe that the dual simplex method is faster than the primal simplex method and the barrier method for classes 4–6 and A–H. On the other hand, the barrier method of CPLEX is faster than the primal and dual simplex methods for RAIL instances. These general purpose LP solvers still require very large computation time and memory space to solve the LP relaxation of huge SCP instances such as RAIL2536–4872.

2.2. Lagrangian relaxation

Another well-known technique is the *Lagrangian relaxation*, which is defined by relaxing some constraints while introducing penalty functions. A typical Lagrangian relaxation problem for SCP is defined as follows:

$$\begin{array}{lllll} (\mathrm{LR}(\boldsymbol{u})) & \text{minimize} & z_{\mathrm{LR}}(\boldsymbol{u}) & = & \sum_{j\in N} c_j x_j + \sum_{i\in M} u_i \left(1 - \sum_{j\in N} a_{ij}x_j\right) \\ & & & = & \sum_{j\in N} \tilde{c}_j(\boldsymbol{u}) x_j + \sum_{i\in M} u_i \\ & \text{subject to} & x_j \in \{0,1\} & & (j \in N), \end{array} \tag{2.2}$$

where $\boldsymbol{u} = (u_1, \ldots, u_m) \in \mathbb{R}_+^m$ is a vector called the *Lagrangian multiplier vector* ($\mathbb{R}_+$ is the set of non-negative real numbers), and $\tilde{c}_j(\boldsymbol{u}) = c_j - \sum_{i\in M} a_{ij}u_i$ is called the *reduced cost* (or *relative cost*) associated with a column $j \in N$. For any $\boldsymbol{u} \in \mathbb{R}_+^m$, $z_{\mathrm{LR}}(\boldsymbol{u})$ gives a lower bound on the optimal value of SCP $z(\boldsymbol{x}^*)$. The problem to find a Lagrangian multiplier vector $\boldsymbol{u} \in \mathbb{R}_+^m$ that maximizes $z_{\mathrm{LR}}(\boldsymbol{u})$ is called the *Lagrangian dual problem*:

$$(\mathrm{LRD}) \qquad \max\left\{ z_{\mathrm{LR}}(\boldsymbol{u}) \mid \boldsymbol{u} \in \mathbb{R}_+^m \right\}. \tag{2.3}$$

For any $\boldsymbol{u} \in \mathbb{R}^m_+$, we can easily obtain an optimal solution to LR($\boldsymbol{u}$), denoted by $\tilde{\boldsymbol{x}}(\boldsymbol{u}) = (\tilde{x}_1(\boldsymbol{u}), \ldots, \tilde{x}_n(\boldsymbol{u}))$, by setting $\tilde{x}_j(\boldsymbol{u}) \leftarrow 1$ if $\tilde{c}_j(\boldsymbol{u}) < 0$ holds and $\tilde{x}_j(\boldsymbol{u}) \leftarrow 0$ if $\tilde{c}_j(\boldsymbol{u}) > 0$ holds, and choosing the value of $\tilde{x}_j(\boldsymbol{u})$ from either zero or one if $\tilde{c}_j(\boldsymbol{u}) = 0$ holds. It is known that any optimal solution to LR($\boldsymbol{u}$) is also optimal for its LP relaxation problem (i.e., replacing the binary constraint $x_j \in \{0,1\}$ with the linear relaxation $0 \leq x_j \leq 1$), which is called the *integrality property*. Hence, the optimal value of the Lagrangian dual problem is equal to that of the LP relaxation problem of SCP [33]. In other words, any optimal solution $\bar{\boldsymbol{u}}$ to the dual of the LP relaxation:

$$\begin{array}{lll} \text{(LPD)} & \text{maximize} & \displaystyle\sum_{i \in M} u_i \\ & \text{subject to} & \displaystyle\sum_{i \in M} a_{ij} u_i \leq c_j \quad (j \in N) \\ & & u_i \geq 0 \qquad\quad\; (i \in M), \end{array} \tag{2.4}$$

is also optimal to the Lagrangian dual problem LRD.

Note that, even if an optimal solution $\tilde{\boldsymbol{x}}(\boldsymbol{u})$ of the Lagrangian relaxation problem LR($\boldsymbol{u}$) is feasible for the original SCP, it is not necessarily optimal for the original SCP. If all constraints are satisfied with equality (i.e., $\sum_{j \in N} a_{ij} \tilde{x}_j(\boldsymbol{u}) = 1$ for all $i \in M$), then $\tilde{\boldsymbol{x}}(\boldsymbol{u})$ is optimal for the original SCP.

We can define another Lagrangian relaxation problem by changing the set of relaxed constraints. For example, Balas and Carrera [3] defined the following Lagrangian relaxation problem:

$$\begin{array}{lll} (\text{LR}'(\boldsymbol{u})) & \text{minimize} & \displaystyle\sum_{j \in N} \left(c_j - \sum_{i \in M \setminus \bar{M}} a_{ij} u_i \right) x_j + \sum_{i \in M \setminus \bar{M}} u_i \\ & \text{subject to} & \displaystyle\sum_{j \in N} a_{ij} x_j \geq 1 \quad (i \in \bar{M}) \\ & & x_j \in \{0,1\} \qquad (j \in N), \end{array} \tag{2.5}$$

where $\bar{M}$ is a maximal subset of M satisfying $N_h \cap N_i = \emptyset$ for all $h, i \in \bar{M}, h \neq i$, and $\boldsymbol{u}$ is a vector indexed by $M \setminus \bar{M}$. They have compared both Lagrangian relaxations experimentally, and found that their Lagrangian relaxation was more robust across different instance classes and converged faster when applying the subgradient optimization, though the difference were not drastic.

2.3. Surrogate relaxation

The *surrogate relaxation* problem is defined by replacing some constraints into a surrogate constraint. The standard surrogate relaxation of SCP is defined as follows:

$$\begin{array}{lll} (\text{S}(\boldsymbol{w})) & \text{minimize} & z_{\text{S}}(\boldsymbol{w}) = \displaystyle\sum_{j \in N} c_j x_j \\ & \text{subject to} & \displaystyle\sum_{i \in M} w_i \left(\sum_{j \in N} a_{ij} x_j \right) \geq \sum_{i \in M} w_i \\ & & x_j \in \{0,1\} \qquad (j \in N), \end{array} \tag{2.6}$$

where $\boldsymbol{w} = (w_1, \ldots, w_m) \in \mathbb{R}^m_+$ is a given vector called the *surrogate multiplier vector*. Compared to Lagrangian relaxation problem, there have been less attempts using the surrogate relaxation problem for integer programming (IP) problems including SCP. Lorena and Lopes [45] proposed a heuristic algorithm for SCP based on a continuous surrogate relaxation.

3. Subgradient Optimization

As discussed in Section 2.2, any optimal solution $\bar{\boldsymbol{u}}$ to the dual of the LP relaxation problem is also an optimal solution to the Lagrangian dual problem; however, computing such $\bar{\boldsymbol{u}}$ by general purpose LP solvers is rather expensive, especially for huge SCP instances as confirmed in Table 2. A common approach to compute a near optimal Lagrangian multiplier vector $\boldsymbol{u}$ is the *subgradient method* [29, 39]. It uses the *subgradient vector* $\boldsymbol{s}(\boldsymbol{u}) = (s_1(\boldsymbol{u}), \ldots, s_m(\boldsymbol{u})) \in \mathbb{R}^m_+$, associated with a given $\boldsymbol{u}$, defined by

$$s_i(\boldsymbol{u}) = 1 - \sum_{j \in N} a_{ij} \tilde{x}_j(\boldsymbol{u}) \qquad (i \in M). \tag{3.1}$$

This method generates a sequence of nonnegative Lagrangian multiplier vectors $\boldsymbol{u}^{(0)}, \boldsymbol{u}^{(1)}, \ldots$, where $\boldsymbol{u}^{(0)}$ is a given initial vector and $\boldsymbol{u}^{(l+1)}$ is updated from $\boldsymbol{u}^{(l)}$ by the following formula:

$$u_i^{(l+1)} \leftarrow \max\left\{ u_i^{(l)} + \lambda \frac{z_{\text{UB}} - z_{\text{LR}}(\boldsymbol{u}^{(l)})}{||\boldsymbol{s}(\boldsymbol{u}^{(l)})||^2} s_i(\boldsymbol{u}^{(l)}), 0 \right\} \qquad (i \in M), \tag{3.2}$$

where z_{UB} is an upper bound on $z(\boldsymbol{x})$, and $\lambda \geq 0$ is a parameter called the step size.

Various implementations of the subgradient method are possible; we therefore start with the following implementation described in Beasley's tutorial [10], and describe several variants afterwards. Let $\boldsymbol{u} = (u_1, \ldots, u_m)$ and $z_{\text{LR}}^{\max}$ be the incumbent Lagrangian multiplier vector and lower bound, respectively. The basic subgradient method is described as follows:

Basic subgradient method (BSM)

Step 1: Set $z_{\text{UB}} \leftarrow z(\boldsymbol{x})$ for a feasible solution $\boldsymbol{x}$ of the original SCP, which is obtained by a greedy algorithm (see Section 6.1). Set $u_i^{(0)} \leftarrow \min\{c_j/|S_j| \mid j \in N_i\}$ and $u_i \leftarrow u_i^{(0)}$ for all $i \in M$ and $z_{\text{LR}}^{\max} \leftarrow z_{\text{LR}}(\boldsymbol{u}^{(0)})$. Set $\lambda \leftarrow 2$ and $l \leftarrow 0$.

Step 2: Compute the current solution $\tilde{\boldsymbol{x}}(\boldsymbol{u}^{(l)})$ and the lower bound $z_{\text{LR}}(\boldsymbol{u}^{(l)})$. If $z_{\text{LR}}(\boldsymbol{u}^{(l)}) > z_{\text{LR}}^{\max}$ holds, set $z_{\text{LR}}^{\max} \leftarrow z_{\text{LR}}(\boldsymbol{u}^{(l)})$ and $u_i \leftarrow u_i^{(l)}$ for all $i \in M$. If $z_{\text{UB}} \leq \lceil z_{\text{LR}}^{\max} \rceil$ holds, output $\boldsymbol{u}$ and $z_{\text{LR}}^{\max}$ and halt (in this case, z_{UB} is optimal for the original SCP).

Step 3: Compute the subgradient vector $\boldsymbol{s}(\boldsymbol{u}^{(l)})$ for the current solution $\tilde{\boldsymbol{x}}(\boldsymbol{u}^{(l)})$. If $s_i(\boldsymbol{u}^{(l)}) = 0$ holds for all $i \in M$, output $\boldsymbol{u}$ and $z_{\text{LR}}^{\max}$ and halt (in this case, $\tilde{\boldsymbol{x}}(\boldsymbol{u}^{(l)})$ is an optimal solution for the original SCP); otherwise compute a new Lagrangian multiplier vector $\boldsymbol{u}^{(l+1)}$ by (3.2).

Step 4: If $z_{\text{LR}}^{\max}$ has not improved in the last 30 iterations with the current value of λ, then set $\lambda \leftarrow 0.5\lambda$. If $\lambda \leq 0.005$ holds, output $\boldsymbol{u}$ and $z_{\text{LR}}^{\max}$ and halt; otherwise let $l \leftarrow l + 1$ and return to Step 2.

The basic subgradient method requires $O(q)$ time for each iteration, where $q = \sum_{i \in M} \sum_{j \in N} a_{ij}$.

Beasley [10] reported a number of detailed observations. For example, the convergence of the subgradient method is relatively insensitive to the initial Lagrangian multiplier vector $\boldsymbol{u}^{(0)}$. Another observation is that it is helpful to set $s_i(\boldsymbol{u}^{(l)}) \leftarrow 0$ when $u_i^{(l)} = 0$ and $s_i(\boldsymbol{u}^{(l)}) < 0$ hold, because $s_i(\boldsymbol{u}^{(l)})^2$ factor reduces the change of $u_h^{(l)}$ $(h \neq i)$.

The subgradient method converges very slowly when the current Lagrangian multiplier vector $\boldsymbol{u}^{(l)}$ approaches almost optimal (i.e., the gap $z_{\text{UB}} - z_{\text{LR}}(\boldsymbol{u}^{(l)})$ gets close to zero). To overcome this, Beasley proposed to replace z_{UB} in (3.2) with $1.05 z_{\text{UB}}$. Caprara et al. [20]

proposed an adaptive control of the step size λ. Their algorithm starts with $\lambda \leftarrow 0.1$. For every 20 iterations, it computes the best and worst lower bounds in the last 20 iterations. If the gap is more than 1%, it decreases the step size by setting $\lambda \leftarrow 0.5\lambda$. On the other hand, if the gap is less than 0.1%, it increases the step size by setting $\lambda \leftarrow 1.5\lambda$. Their algorithm halts when the improvement of the best lower bound $z_{\mathrm{LR}}^{\max}$ in the last 300 iterations is less than 0.1% and 1.0 in value. We call this rule to update λ adaptively rule ASSC (abbreviation for adaptive step size control), and use it in our computational experiments later.

Caprara et al. [20] also dealt with slow convergence due to the degeneracy of the Lagrangian relaxation problem, which often appears for huge SCP instances with relatively uniform costs. In such a situation, after a few subgradient iterations, a large number of columns $j \in N$ happens to have reduced costs $\tilde{c}_j(\boldsymbol{u}^{(l)})$ to be very close to zero, and consequently the subgradient vector $\boldsymbol{s}(\boldsymbol{u}^{(l)})$ highly oscillates in the subsequent iterations. To overcome this, they proposed a heuristic rule to define a vector $\boldsymbol{s}(\boldsymbol{u}^{(l)})$ as follows:

Heuristic computation of the subgradient vector (HCSV)

Step 1: Set $\tilde{N} \leftarrow \{j \in N \mid \tilde{c}_j(\boldsymbol{u}^{(l)}) \leq 0.001\}$ and $\tilde{M} \leftarrow \bigcup_{j \in \tilde{N}} S_j$.

Step 2: Let $R \leftarrow \{j \in \tilde{N} \mid \bigcup_{k \in \tilde{N}\setminus\{j\}} S_k = \tilde{M}\}$, which is the set of redundant columns. Sort all columns $j \in R$ in the descending order of their reduced cost $\tilde{c}_j(\boldsymbol{u}^{(l)})$.

Step 3: For all $j \in R$, according to the above order, set $\tilde{N} \leftarrow \tilde{N}\setminus\{j\}$ if $\bigcup_{k \in \tilde{N}\setminus\{j\}} S_k = \tilde{M}$ holds.

Step 4: Set $\hat{x}_j \leftarrow 1$ for all $j \in \tilde{N}$ and set $\hat{x}_j \leftarrow 0$ for all $j \notin \tilde{N}$. Set $s_i(\boldsymbol{u}^{(l)}) = 1 - \sum_{j \in N} a_{ij}\hat{x}_j$ for all $i \in M$.

This procedure requires $O(n \log n)$ time for sorting the redundant columns, plus $O(q)$ time for the remaining computation. Note that the vector $\boldsymbol{s}(\boldsymbol{u}^{(l)})$ obtained by the above heuristic rule is no longer guaranteed to be a subgradient vector.

Balas and Carrera [3] proposed a simple procedure that transforms the Lagrangian multiplier vector $\boldsymbol{u}$ into a dual feasible solution of the LP relaxation problem without decreasing (and possibly increasing) the associated lower bound $z_{\mathrm{LR}}(\boldsymbol{u})$. The procedure is shown as follows.

Transformation of the Lagrangian multiplier vector (TLMV)

Step 1: If there exists a column $j \in N$ with $\tilde{c}_j(\boldsymbol{u}) < 0$, select a row $i \in S_j$ with $u_i > 0$. Otherwise go to Step 3.

Step 2: If $u_i < |\tilde{c}_j(\boldsymbol{u})|$ holds, set $u_i \leftarrow 0$; otherwise set $u_i \leftarrow u_i + \tilde{c}_j(\boldsymbol{u})$. Return to Step 1.

Step 3: Set $\tilde{x}_j(\boldsymbol{u}) \leftarrow 1$ if $\tilde{c}_j(\boldsymbol{u}) = 0$ holds and $\tilde{x}_j(\boldsymbol{u}) \leftarrow 0$ if $\tilde{c}_j(\boldsymbol{u}) > 0$ holds for all $j \in N$. If there exists no uncovered row $i \in M$ by the current solution $\tilde{\boldsymbol{x}}(\boldsymbol{u})$, output $\boldsymbol{u}$ and halt. Otherwise select an uncovered row $i \in M$ and set $u_i \leftarrow u_i + \min_{j \in N_i} \tilde{c}_j(\boldsymbol{u})$, and then return to Step 3.

This procedure requires $O(q)$ time. It is not hard to observe that $z_{\mathrm{LR}}(\boldsymbol{u})$ never decreases whenever $\boldsymbol{u}$ is updated. The multiplier vector $\boldsymbol{u}$ is feasible to LPD when the algorithm proceeds from Step 1 to Step 3, and remains feasible during the execution of Step 3. This method can be viewed as a hybrid approach of *multiplier adjustment* (Steps 1 and 2) and *dual ascent* (Step 3) methods, whose general ideas are summarized in [10].

Table 3: Lower bounds obtained by the subgradient methods BSM, MSM1 and MSM2

Instance	Rows	Columns	z_{LP}	Lower bounds		
				BSM	MSM1	MSM2
4.1–4.10	200	1000	509.10	509.03	508.79	509.06
5.1–5.10	200	2000	256.38	256.32	256.03	256.34
6.1–6.5	200	1000	139.22	139.02	137.00	139.04
A.1–A.5	300	3000	237.73	237.55	236.65	237.58
B.1–B.5	300	3000	69.38	69.26	65.92	69.25
C.1–C.5	400	4000	219.34	219.15	217.86	219.20
D.1–D.5	400	4000	58.84	58.72	53.76	58.72
E.1–E.5	500	5000	21.38	21.26	17.51	21.27
F.1–F.5	500	5000	8.92	8.79	7.03	8.78
G.1–G.5	1000	10,000	149.48	149.13	139.45	149.17
H.1–H.5	1000	10,000	45.67	45.41	38.01	45.44
RAIL507	507	63,009	172.15	171.41	170.77	171.68
RAIL516	516	47,311	182.00	181.53	180.58	181.63
RAIL582	582	55,515	209.71	209.40	208.48	209.49
RAIL2536	2536	1,081,841	688.40	685.79	682.30	686.65
RAIL2586	2586	920,683	935.92	933.19	928.32	934.05
RAIL4284	4284	1,092,610	1054.05	1051.70	1043.04	1052.57
RAIL4872	4872	968,672	1509.64	1505.86	1493.83	1507.71

Ceria et al. [24] proposed a primal-dual subgradient method, which generates a pair of Lagrangian multiplier vectors $(\boldsymbol{x}, \boldsymbol{u})$, and approaches the optimal value of the LP relaxation problem from both lower and upper sides. To this end, they first defined a Lagrangian relaxation problem to the LP dual problem (2.4) by introducing a Lagrangian multiplier vector $\boldsymbol{x} = (x_1, \ldots, x_n)$. We can define a Lagrangian dual problem optimizing the Lagrangian multiplier vector $\boldsymbol{x}$, which is equivalent to the LP relaxation problem for SCP. Accordingly, they proposed a subgradient method, which updates both Lagrangian multiplier vectors $\boldsymbol{x}$ and $\boldsymbol{u}$ simultaneously.

We tested the basic subgradient method (BSM) and two modified subgradient methods. The first modified subgradient method (denoted MSM1) uses the rule ASSC to control the step size λ and the heuristic algorithm HCSV to compute the subgradient vector $\boldsymbol{s}(\boldsymbol{u})$. To be more precise, rules of BSM are modified as follows: (i) The rule to initialize λ in Step 1, the rule to update λ and the stopping criterion in Step 4 are replaced with rule ASSC; (ii) the vector obtained by algorithm HCSV is used instead of the subgradient vector $\boldsymbol{s}(\boldsymbol{u}^{(l)})$ in Step 3. The second modified version (MSM2) uses the algorithm TLMV to improve the lower bound at every iteration. More precisely, rules of BSM are modified as follows: Let $\boldsymbol{u}'$ be the vector obtained by applying algorithm TLMV to the current multiplier vector $\boldsymbol{u}^{(l)}$, and then use $\boldsymbol{u}'$ instead of $\boldsymbol{u}^{(l)}$ in Step 2. The other parts of algorithm MSM1 and MSM2 are exactly the same as BSM. It might seem more natural to use this modified multiplier vector $\boldsymbol{u}'$ also in Step 3 to compute the next multiplier vector $\boldsymbol{u}^{(l+1)}$; however, we observed through preliminary experiments that the lower bounds obtained with this option is worse than those of BSM.

Tables 3 and 4 show the lower bounds, the number of iterations (column "iter.") and the computation time in seconds of BSM, MSM1 and MSM2. From Tables 3 and 4, we can observe that these subgradient methods obtain near optimal lower bounds quickly in spite of their simplicity. We can also observe that the lower bounds of MSM1 are worse than BSM for all instances, and those of MSM2 are slightly better than BSM, though MSM2 consumes

Table 4: Number of iterations and computation time in seconds of the subgradient methods BSM, MSM1 and MSM2

Instance	Rows	Columns	BSM		MSM1		MSM2	
			Iter.	Time	Iter.	Time	Iter.	Time
4.1–4.10	200	1000	536.3	0.03	2231.8	0.18	557.3	0.07
5.1–5.10	200	2000	610.1	0.06	1963.2	0.26	596.1	0.11
6.1–6.5	200	1000	831.2	0.09	3382.8	0.49	767.6	0.15
A.1–A.5	300	3000	751.8	0.15	2740.8	0.69	762.4	0.28
B.1–B.5	300	3000	1022.8	0.47	3104.4	1.72	992.4	0.70
C.1–C.5	400	4000	918.6	0.32	3440.0	1.47	939.2	0.54
D.1–D.5	400	4000	1048.4	0.80	2519.2	2.41	1023.8	1.23
E.1–E.5	500	5000	1399.8	3.31	2144.8	6.57	1297.2	5.27
F.1–F.5	500	5000	1430.2	7.13	1714.6	10.74	1371.0	11.88
G.1–G.5	1000	10,000	1236.6	2.38	4041.8	10.26	1188.2	3.83
H.1–H.5	1000	10,000	1478.2	7.02	2404.2	15.43	1548.0	13.08
RAIL507	507	63,009	698	3.06	969	5.28	867	8.64
RAIL516	516	47,311	780	2.58	798	3.23	848	7.20
RAIL582	582	55,515	1009	4.34	927	4.95	802	9.27
RAIL2536	2536	1,081,841	1034	111.44	2184	374.17	972	410.72
RAIL2586	2586	920,683	892	73.17	1306	152.95	845	238.36
RAIL4284	4284	1,092,610	1239	135.52	1807	314.24	1190	485.38
RAIL4872	4872	968,672	964	89.20	1351	182.86	1145	351.91

more computation time than BSM. To see their convergence properties, we illustrate their behavior in Figures 1 and 2, where we imposed no limit on the number of their iterations. From Figures 1 and 2, we can observe that MSM1 and MSM2 obtain comparable lower bounds with smaller numbers of iterations than BSM.

4. Sifting Method

In order to solve huge LP relaxation problem, Bixby et al. [15] developed a variant of the column generation method called the *sifting method* (or *pricing method*), which generates successive solutions of small subproblems by taking a small subset of columns $C \subset N$.

The sifting method is based on the observation that only a small number of columns $j \in N$ with negative reduced costs $\tilde{c}_j(\boldsymbol{u}) < 0$ are necessary to compute the current objective value z_{LP} at every iteration of the simplex method. It will therefore be advantageous to solve a number of subproblems called the *core problem* consisting of a small subset C of columns j with small reduced costs $\tilde{c}_j(\boldsymbol{u})$ and update the current core problem at moderate intervals.

Caprara et al. [20] developed a sifting procedure on the subgradient method for SCP. The initial core problem C is defined by taking the columns $j \in N_i$ with the five smallest values of reduced costs $\tilde{c}_j(\boldsymbol{u})$ for each row $i \in M$, and the current core problem C is updated every T subgradient iterations. The next core problem C is mainly composed of columns $j \in N$ with smallest reduced costs $\tilde{c}_j(\boldsymbol{u})$ for the current Lagrangian multiplier vector $\boldsymbol{u}$. The rule to update the core problem C is formally described as follows.

Updating core problem

Step 1: Compute the reduced cost $\tilde{c}_j(\boldsymbol{u})$ for all $j \in N$.

Step 2: Set $C_1 \leftarrow \{j \in N \mid \tilde{c}_j(\boldsymbol{u}) < 0.1\}$. For each $i \in M$, let $C_2(i)$ be the columns with the five smallest values of $\tilde{c}_j(\boldsymbol{u})$ among those in N_i. Then set $C_2 \leftarrow \bigcup_{i \in M} C_2(i)$.

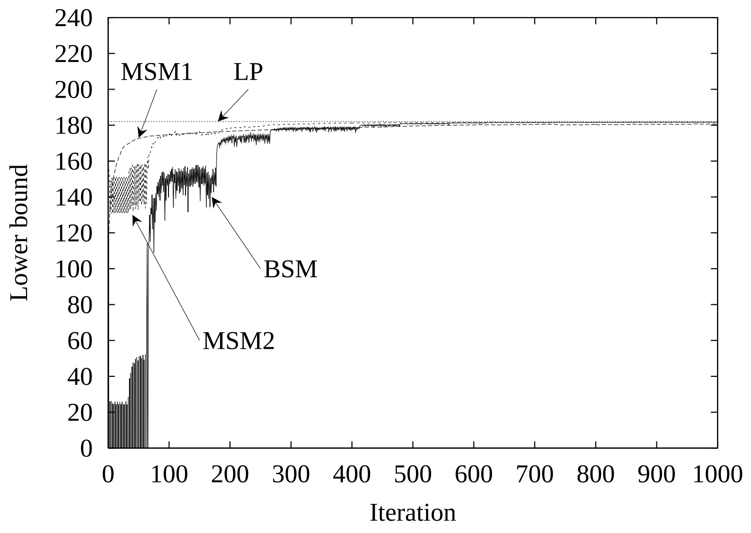

Figure 1: Comparison of the subgradient methods BSM, MSM1 and MSM2 on instance RAIL516

Step 3: If $|C_1| > 5m$ holds, let C_1 be the set of columns $j \in C_1$ having the $5m$ smallest values of $\tilde{c}_j(\boldsymbol{u})$. Set $C \leftarrow C_1 \cup C_2$.

We note that the optimal value for the core problem $z^c_{\text{LR}}(\boldsymbol{u}) = \sum_{j \in C} \tilde{c}_j(\boldsymbol{u})\tilde{x}_j(\boldsymbol{u}) + \sum_{i \in M} u_i$ does not necessarily give a lower bound for the original SCP and we need to compute the lower bound $z_{\text{LR}}(\boldsymbol{u})$ by the following formula:

$$z_{\text{LR}}(\boldsymbol{u}) = z^c_{\text{LR}}(\boldsymbol{u}) + \sum_{j \in N \setminus C} \tilde{c}_j(\boldsymbol{u})\tilde{x}_j(\boldsymbol{u}) = \sum_{i \in M} u_i + \sum_{j \in N} \min\{\tilde{c}_j(\boldsymbol{u}), 0\}. \tag{4.1}$$

One of the most important decisions in implementing the sifting method is how to control the updating interval T. If the sifting method updates the core problem C rarely, the objective value $z^c_{\text{LR}}(\boldsymbol{u})$ of the core problem becomes far from the lower bound $z_{\text{LR}}(\boldsymbol{u})$ obtained by the original Lagrangian relaxation problem. On the other hand, if the sifting method updates the core problem C frequently, it consumes much computation time for updating the core problem C and computing the lower bound $z_{\text{LR}}(\boldsymbol{u})$ of the original Lagrangian relaxation problem. They proposed the following sophisticated rule to control the updating interval T.

They first set $T \leftarrow 10$ and compute the relative gap $\Delta = (z^c_{\text{LR}}(\boldsymbol{u}) - z_{\text{LR}}(\boldsymbol{u}))/z_{\text{UB}}$ after each updating, where z_{UB} is the best upper bound obtained by then. If Δ is small, they increase the updating interval T. On the other hand, if Δ is large, they decrease the updating interval T. More precisely, they set

$$T \leftarrow \begin{cases} 10T & \Delta \le 10^{-6} \\ 5T & 10^{-6} < \Delta \le 0.02 \\ 2T & 0.02 < \Delta \le 0.2 \\ 10 & \Delta > 0.2. \end{cases} \tag{4.2}$$

Table 5 shows lower bounds and computation time in seconds of two sifting methods, where one sifting method is implemented on the basic subgradient method (BSM) and the other sifting method is a component of CPLEX 9.1.3 implemented on the dual simplex

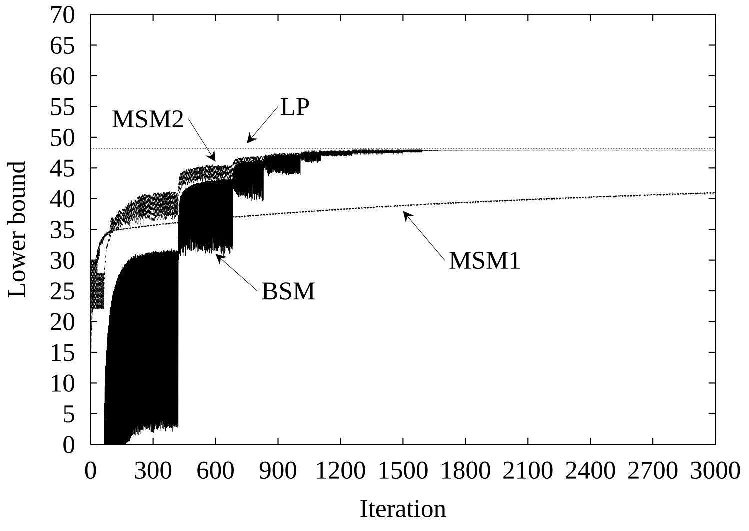

Figure 2: Comparison of the subgradient methods BSM, MSM1 and MSM2 on instance H.1

method. From Table 5, we can see that these sifting methods reduce much computation time without sacrificing the solution quality. Note that, if the sifting method of CPLEX 9.1.3 generates a feasible solution $\boldsymbol{u}$ of LPD, it can be used to compute a lower bound of the original SCP. We observed that the lower bounds obtained in this way was very good even with small number of iterations, though the time to compute an exact optimum is more expensive than BSM.

5. Problem Reduction

There are many procedures in the literature that test feasibility and reduce the size of an SCP instance by removing redundant rows and columns [7, 21, 30, 32, 43]. Common rules for problem reduction are described as follows:

Rule 1: If $|N_i| = 0$ holds for at least one row $i \in M$, the instance is infeasible.

Rule 2: If $|N_i| = 1$ holds for some rows $i \in M$, for every such row i, set $x_j \leftarrow 1$ for the unique column $j \in N_i$, and remove all rows $k \in S_j$ and the column j.

Rule 3: If there exists a pair of rows $h, i \in M$ such that $N_h \subseteq N_i$ holds, remove the row i.

Rule 4: If there exists a column $j \in N$ and a set of columns $N' \subseteq N \setminus \{j\}$ such that $S_j \subseteq (\bigcup_{k \in N'} S_k)$ and $c_j \geq \sum_{k \in N'} c_k$ hold, set $x_j \leftarrow 0$ and remove the column j.

These rules can be applied repeatedly until infeasibility is proven or no rule becomes applicable. If naively implemented, the time complexity of Rules 1–4 are $O(m)$, $O(q)$, $O(m^2 n)$ and $O(qn2^{n-1})$, respectively. Since a naive implementation of these rules is quite time consuming, they have to be implemented in a careful way, or substituted by other simple rules. For example, Rule 4 is often replaced with the following rule [7, 21, 30]:

Rule 4′: If there exists a column $j \in N$ such that (i) $c_j > \sum_{i \in S_j} \min_{k \in N_i} c_k$ or (ii) $|S_j| > 1$ and $c_j \geq \sum_{i \in S_j} \min_{k \in N_i} c_k$ hold, then set $x_j \leftarrow 0$ and remove the column j.

Table 5: Computational results of the sifting methods

Instance	Rows	Columns	BSM		CPLEX9.1.3	
			z_{LR}	Time	z_{LP}	Time
4.1–4.10	200	1000	508.99	0.07	509.10	0.05
5.1–5.10	200	2000	256.07	0.04	256.38	0.05
6.1–6.5	200	1000	139.07	0.20	139.22	0.07
A.1–A.5	300	3000	237.64	0.13	237.73	0.11
B.1–B.5	300	3000	69.30	0.14	69.38	0.15
C.1–C.5	400	4000	219.22	0.14	219.34	0.22
D.1–D.5	400	4000	58.68	0.15	58.84	0.27
E.1–E.5	500	5000	21.12	0.27	21.38	0.61
F.1–F.5	500	5000	8.61	0.43	8.92	0.90
G.1–G.5	1000	10,000	149.11	0.40	149.48	0.80
H.1–H.5	1000	10,000	45.17	0.56	45.67	0.74
RAIL507	507	63,009	171.53	1.56	172.15	1.94
RAIL516	516	47,311	181.67	0.95	182.00	0.66
RAIL582	582	55,515	209.35	1.45	209.71	1.63
RAIL2536	2536	1,081,841	684.79	37.14	688.40	149.16
RAIL2586	2586	920,683	923.10	18.02	935.92	82.73
RAIL4284	4284	1,092,610	1051.48	38.73	1054.05	610.80
RAIL4872	4872	968,672	1506.90	26.66	1509.64	241.86

The conditions (i) and (ii) are designed so that they do not remove those columns j that attain the minimum for at least one row (i.e., $\exists i,\ c_j = \min_{k \in N_i} c_k$); the strict inequality in (i) and the condition $|S_j| > 1$ in (ii) are necessary for this purpose. The time complexity of this rule is $O(q)$ if appropriately implemented.

These rules are usually used for preprocessing, but can be applied at each iteration of heuristic algorithms or each node of branch-and-bound algorithms after some variables are fixed to zero or one. However, since we need much computational effort to reflect the removal of rows and columns on the matrix (a_{ij}), most algorithms use a part of the problem reduction rules only for preprocessing.

We tested the problem reduction rules for the benchmark instances. Since we observed that Rule 3 is not very effective and more time consuming than other rules, we replaced Rule 3 with the following rule:

Rule 3$'$: If there exists a pair of rows $h, i \in M$ such that $N_h = N_i$ holds, remove the row i.

The time complexity of this rule is $O(mn)$ if appropriately implemented.

We also adopt Rule 4$'$ instead of Rule 4 because it is very expensive to check Rule 4 throughly. Since we observed through preliminary experiments that the conditions (i) and (ii) in Rule 4$'$ gave almost the same results, we apply both of them in Rule 4$'$. We apply Rules 1, 2, 3$'$ and 4$'$ repeatedly until no rule is applicable or the infeasibility of the instance is detected. Table 6 reports the size of reduced instances, frequency of each rule (i.e., how many times each rule transforms the instance) and computation time in seconds of the problem reduction procedure.

We first observe that only Rule 4$'$ works effectively. However, we also observe that Rule 4$'$ works less effectively for classes E and H, and has very little effect for classes F and RAIL. This is because each column $j \in N$ covers relatively many rows $i \in M$ in classes E, F and H and all costs are distributed in a narrow range in RAIL instances.

Table 6: Computational results of the problem reduction procedure

Instance	Original size		Reduced size		Frequency			Time
	Rows	Columns	Rows	Columns	R2	R3′	R4′	
4.1–4.10	200	1000	177.7	191.2	5.5	0.0	805.0	0.01
5.1–5.10	200	2000	177.5	192.6	5.1	0.0	1801.6	0.01
6.1–6.5	200	1000	200.0	244.8	0.0	0.0	755.2	0.02
A.1–A.5	300	3000	300.0	372.4	0.0	0.0	2627.6	0.01
B.1–B.5	300	3000	300.0	552.4	0.0	0.0	2447.6	0.02
C.1–C.5	400	4000	400.0	544.4	0.0	0.0	3455.6	0.04
D.1–D.5	400	4000	400.0	838.8	0.0	0.0	3161.2	0.02
E.1–E.5	500	5000	500.0	2467.2	0.0	0.0	2532.8	0.02
F.1–F.5	500	5000	500.0	4782.8	0.0	0.0	217.2	0.01
G.1–G.5	1000	10,000	1000.0	2175.8	0.0	0.0	7824.2	0.06
H.1–H.5	1000	10,000	1000.0	4930.0	0.0	0.0	5070.0	0.05
RAIL507	507	63,009	482	62,991	8	5	9	0.02
RAIL516	516	47,311	445	47,266	24	17	17	0.01
RAIL582	582	55,515	566	55,404	6	3	105	0.02
RAIL2536	2536	1,081,841	2486	1,081,822	5	18	12	0.28
RAIL2586	2586	920,683	2467	920,196	33	25	445	0.33
RAIL4284	4284	1,092,610	4176	1,092,191	14	49	404	0.30
RAIL4872	4872	968,672	4663	968,397	51	32	210	0.25

From these results, it is effective to apply the problem reduction procedure with Rules 1, 2, 3′ and 4′ since its computation time is short. However, taking account of the trade-off between implementation effort and efficiency, it is worthwhile to apply the reduction procedure only with Rule 4′ only if an SCP instance has low density and widely distributed costs.

Another type of problem reduction rules are derived from the reduced cost $\tilde{c}_j(\boldsymbol{u})$ of the Lagrangian relaxation problem. We are given a solution $\tilde{\boldsymbol{x}}(\boldsymbol{u}) = (\tilde{x}_1(\boldsymbol{u}), \ldots, \tilde{x}_n(\boldsymbol{u}))$ of the Lagrangian relaxation problem LR($\boldsymbol{u}$). If we impose an additional constraint $x_j = 1$ for a particular column $j \in N$ with $\tilde{x}_j(\boldsymbol{u}) = 0$, then we obtain a better lower bound $z_{\text{LR}}(\boldsymbol{u}) + \tilde{c}_j(\boldsymbol{u})$. Similarly, if we impose an additional constraint $x_j = 0$ for a particular column $j \in N$ with $\tilde{x}_j(\boldsymbol{u}) = 1$ then we obtain a better lower bound $z_{\text{LR}}(\boldsymbol{u}) - \tilde{c}_j(\boldsymbol{u})$ (recall that $\tilde{x}_j(\boldsymbol{u})$ takes one when $\tilde{c}_j(\boldsymbol{u})$ is negative). Accordingly, we can describe the following problem reduction rules:

Rule 5: If $\tilde{x}_j(\boldsymbol{u}) = 0$ and $z_{\text{LR}}(\boldsymbol{u}) + \tilde{c}_j(\boldsymbol{u}) > z_{\text{UB}}$ hold, set $x_j \leftarrow 0$ and remove the column j.

Rule 6: If $\tilde{x}_j(\boldsymbol{u}) = 1$ and $z_{\text{LR}}(\boldsymbol{u}) - \tilde{c}_j(\boldsymbol{u}) > z_{\text{UB}}$ hold, set $x_j \leftarrow 1$, and remove all rows $i \in S_j$ and the column j.

These techniques have often been used in branch-and-bound algorithms and are called the *variable fixing* (or *pegging test*).

Balas and Carrera [3] proposed an improved variable fixing technique. They first define a new Lagrangian relaxation problem by the removal of the covered rows $i \in S_j$ according to fixing a variable x_j to one for a particular column $j \in N$. This is done by setting $u_i \leftarrow 0$ for all $i \in S_j$ on the original Lagrangian relaxation problem. Then, they recompute the reduced costs $\tilde{c}_j(\boldsymbol{u})$ for all columns $j \in N_i$ and apply Rule 5.

We note that these variable fixing techniques for a variable x_j never work when the gap between the upper bound z_{UB} and the lower bound $z_{\text{LR}}(\boldsymbol{u})$ is larger than c_j. We also note that it is often complicated or quite time consuming to change the data structure of the

Table 7: Computational results of the variable fixing

Instance	Original size		Reduced size		Frequency		Time
	Rows	Columns	Rows	Columns	R5	R6	
4.1–4.10	200	1000	199.2	293.8	706.1	0.1	0.05
5.1–5.10	200	2000	198.1	381.9	1617.7	0.4	0.09
6.1–6.5	200	1000	200.0	251.8	748.2	0.0	0.15
A.1–A.5	300	3000	300.0	606.0	2394.0	0.0	0.23
B.1–B.5	300	3000	300.0	357.4	2642.6	0.0	0.59
C.1–C.5	400	4000	400.0	772.0	3228.0	0.0	0.48
D.1–D.5	400	4000	400.0	567.8	3432.2	0.0	1.16
E.1–E.5	500	5000	500.0	573.2	4426.8	0.0	4.09
F.1–F.5	500	5000	500.0	428.4	4571.6	0.0	8.97
G.1–G.5	1000	10,000	1000.0	3186.8	6813.2	0.0	3.19
H.1–H.5	1000	10,000	1000.0	2304.6	7695.4	0.0	9.44
RAIL507	507	63,009	507	63,009	0	0	6.06
RAIL516	516	47,311	516	47,311	0	0	3.70
RAIL582	582	55,515	582	55,515	0	0	6.33
RAIL2536	2536	1,081,841	2536	1,081,841	0	0	160.24
RAIL2586	2586	920,683	2586	920,683	0	0	117.89
RAIL4284	4284	1,092,610	4284	1,092,610	0	0	223.16
RAIL4872	4872	968,672	4872	968,672	0	0	160.17

SCP instance when some reduction rules are successfully applied. Beasley [10] reported that it is computationally worthwhile to reflect the removal of rows and columns on the matrix (a_{ij}) when a significant amount of reduction have been achieved, e.g., 10% of variables are fixed to zero or one.

Table 7 shows the size of reduced instances, frequency of each rule and computation time in seconds of the basic subgradient method (BSM) including the time for checking Rules 5 and 6, which are applied at every iteration. From Table 7, we observe that Rule 5 fixes many variables with little computational effort for classes 4–6 and A–H while fixing no variable for RAIL instances. From this, it is worth applying Rule 5 if an SCP instance has widely distributed costs. We also illustrate the number of fixed variables at every iteration in Figure 3. We can observe that BSM with Rules 5 and 6 fixes many variables with a small number of iterations; the number of fixed variables reaches 5000 very quickly, and becomes more than 7000 after a few hundred iterations.

6. Heuristic Algorithms

6.1. Construction algorithms

Several construction algorithms with performance guarantee have been developed for SCP. These construction algorithms are not only interesting in theoretical aspect but contribute to develop efficient heuristic algorithms in practical applications. In this section, we explain five representative construction algorithms and compare their experimental performance.

One of the most representative construction algorithms for SCP is the *greedy algorithm*, which iteratively selects the most cost effective column $j \in N$ until all rows $i \in M$ are covered.

```
Greedy algorithm
Step 1: Set $M' \leftarrow \emptyset$ and $x_j \leftarrow 0$ for all $j \in N$.
Step 2: If $M' = M$ holds, go to Step 3. Otherwise, find a column $j \in N$ with the
```

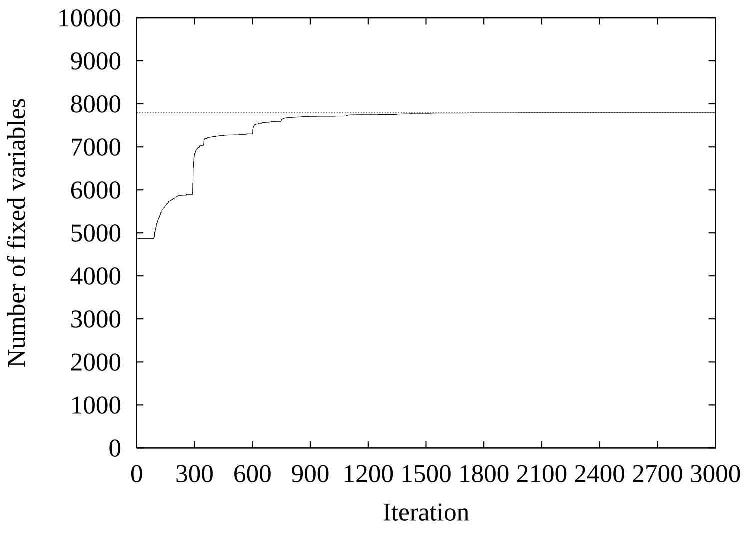

Figure 3: Number of fixed variables at every iteration on instance H.1

minimum cost effectiveness $c_j/|S_j \setminus M'|$ among those with $x_j = 0$, set $x_j \leftarrow 1$ and $M' \leftarrow M' \cup S_j$, and return to Step 2.

Step 3: If there exists a redundant column $j \in N$ (i.e., $x_j = 1$ and $\sum_{j' \in N \setminus \{j\}} a_{ij'} x_{j'} = 1$ for all $i \in M$), set $x_j \leftarrow 0$ and return to Step 3; otherwise output $\boldsymbol{x}$ and halt.

In the above algorithm, redundant columns are usually removed in the reverse order of selecting columns, which is called the *reverse delete step*. The greedy algorithm is known to be an H_n-approximation algorithm [25], where H_n is called the Harmonic series defined as follows

$$H_n = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n},$$

and is roughly estimated by $O(\log n)$. Ho [40] showed that all variants of the greedy algorithm on the cost effectiveness functions

$$c_j, \frac{c_j}{|S_j \setminus M'|}, \frac{c_j}{\log_2 |S_j \setminus M'|}, \frac{c_j}{|S_j \setminus M'| \log_2 |S_j \setminus M'|}, \frac{c_j}{|S_j \setminus M'| \ln |S_j \setminus M'|}, \frac{c_j}{|S_j \setminus M'|^2}, \frac{(c_j)^{1/2}}{|S_j \setminus M'|^2}$$

have the same worst case behavior. Balas and Ho [4] and Vasko and Wilson [47] showed experimental performance of the greedy algorithm with various cost effectiveness functions.

Although a naive implementation of the greedy algorithm requires $O(mn)$ time, Caprara et al. [20] developed $O(rn + q)$ time implementation of the greedy algorithm by devising an efficient procedure to update cost effectiveness, where $r = \sum_{j \in N} x_j$. Since the term rn, which is the total time for finding the most cost effective column, is typically much larger than q for SCP instances of low density, they also developed an efficient implementation to reduce the practical computation time of this part substantially (though its worst case time complexity is the same).

Another natural algorithm is obtained by rounding an optimal solution $\bar{\boldsymbol{x}} = (\bar{x}_1, \dots, \bar{x}_n)$ of the LP relaxation problem into an integer solution $\boldsymbol{x} = (x_1, \dots, x_n)$. We first define $f = \max_{i \in M} |N_i|$, which is the frequency of the most frequent column. The *rounding algorithm* for SCP gives an integer solution $\boldsymbol{x} = (x_1, \dots, x_n)$ as follows:

$$x_j \leftarrow \begin{cases} 1 & \bar{x}_j \geq 1/f \\ 0 & \text{otherwise.} \end{cases} \tag{6.1}$$

The solution $\boldsymbol{x}$ is feasible for the original SCP (i.e., $\sum_{j\in N} a_{ij}x_j \geq 1$ for all $i \in M$) since $\sum_{j\in N} a_{ij}\bar{x}_j \geq 1$ for each $i \in M$ and hence at least one $j \in N_i$ must satisfy $\bar{x}_j \geq 1/|N_i| \geq 1/f$. The rounding algorithm requires $O(n)$ time in addition to the computation time for solving the LP relaxation problem, and is known to be an f-approximation algorithm [48]. Note that, since we have $|N_i| = 2$ for all rows $i \in M$ in the vertex cover problem (VCP), any f-approximation algorithm for SCP is a 2-approximation algorithm for VCP.

The *primal-dual method* is one of the most representative approaches for designing approximation algorithms. It is based on the following characterization of optimal solutions $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{u}})$ to a pair of primal and dual LP problems, called the primal and dual *complementary slackness conditions*, respectively:

$$\bar{x}_j > 0 \;\Rightarrow\; \sum_{i\in M} a_{ij}\bar{u}_i = c_j \qquad (j \in N), \tag{6.2}$$

$$\bar{u}_i > 0 \;\Rightarrow\; \sum_{j\in N} a_{ij}\bar{x}_j = 1 \qquad (i \in M), \tag{6.3}$$

where (6.2) is also written as $\bar{x}_j > 0 \Rightarrow \tilde{c}_j(\boldsymbol{u}) = 0$.

Hochbaum [41] proposed a simple primal-dual algorithm. It first finds an optimal solution $\bar{\boldsymbol{u}}$ of the dual LP relaxation problem, and set $x_j \leftarrow 1$ for all $j \in N$ satisfying $\sum_{i\in M} a_{ij}\bar{u}_i = c_j$ and $x_j \leftarrow 0$ for the others. To see $\boldsymbol{x}$ is feasible to the original SCP, observe that, if there exists a row i' that satisfies $\sum_{i\in M} a_{ij}\bar{u}_i < c_j$ for all $j \in N_{i'}$, then we can increase $\bar{u}_{i'}$ slightly without violating the dual feasibility, which contradicts with the optimality of $\bar{\boldsymbol{u}}$. The running time of this algorithm is $O(n)$ in addition to the computation time for solving the LP relaxation problem. Since the solution $\boldsymbol{x}$ obtained by this algorithm satisfies the primal complementary slackness condition, we observe that

$$\sum_{j\in N} c_j x_j = \sum_{j\in N}\left(\sum_{i\in M} a_{ij}\bar{u}_i\right) x_j = \sum_{i\in M}\left(\sum_{j\in N} a_{ij}x_j\right)\bar{u}_i \leq \max_{i\in M}|N_i| \sum_{i\in M}\bar{u}_i \leq f\cdot z_{\mathrm{LP}}. \tag{6.4}$$

Hence, this algorithm is an f-approximation algorithm.

A typical primal-dual algorithm starts from a pair of primal infeasible solution $\boldsymbol{x}$ and dual feasible solution $\boldsymbol{u}$. It iteratively modifies the current solution $(\boldsymbol{x}, \boldsymbol{u})$ to improve primal feasibility and dual optimality while satisfying the primal complementary slackness condition (6.2) until a primal feasible solution is obtained. During the iterations, the primal solution $\boldsymbol{x}$ is always modified integrally, so that eventually we obtain an integer solution. We describe a basic primal-dual algorithm for SCP by Bar-Yehuda and Even [6].

Basic primal-dual algorithm

Step 1: Set $x_j \leftarrow 0$ for all $j \in N$ and $u_i \leftarrow 0$ for all $i \in M$.

Step 2: If there exists no uncovered row $i \in M$, go to Step 3. Otherwise select an uncovered row $i \in M$ and set

$$u_i \leftarrow \min_{j\in N_i}\left\{ c_j - \sum_{h\in M, h\neq i} a_{hj}u_h \right\}.$$

Then, set $x_j \leftarrow 1$ for a column $j \in N_i$ satisfying $\sum_{h\in M} a_{hj}u_h = c_j$ and return to Step 2.

Step 3: If there exists a redundant column $j \in N$, set $x_j \leftarrow 0$ and return to Step 3; otherwise output $\boldsymbol{x}$ and halt.

Note that we do not need to solve LP relaxation problems though the algorithm is based on the properties of LP relaxation. The basic primal-dual algorithm can be easily implemented with $O(q)$ time. Since the above analysis on the performance guarantee (6.4) is also applicable to the basic primal-dual algorithm by replacing $\bar{\boldsymbol{u}}$ with the $\boldsymbol{u}$ obtained by the algorithm, it is also an f-approximation algorithm.

Agrawal et al. [1] developed an improved primal-dual algorithm for network design problems, which increases simultaneously and at the same speed all the dual variables that can be increased without violating the dual feasibility. They also showed the primal-dual algorithms with such uniform increase rule have better performance guarantees for several network design problems than the basic primal-dual algorithms [1, 34]. Goemans and Williamson [35] applied this method to SCP and showed detailed analysis of its performance guarantee.

Let M' be the set of covered rows $i \in M$ and Δ be the increase of the dual variables u_i for uncovered rows $i \in M \setminus M'$ at each iteration. When dual variables u_i for all $i \in M \setminus M'$ are increased by Δ, the reduced cost $\tilde{c}_j(\boldsymbol{u})$ for each $j \in N$ decreases $|S_j \setminus M'|\Delta$. From this, Δ at each iteration is computed by the following formula:

$$\Delta \leftarrow \min_{j \in N_i, i \notin M'} \frac{\tilde{c}_j(\boldsymbol{u})}{|S_j \setminus M'|}. \tag{6.5}$$

Primal-dual algorithm with uniform increase rule

Step 1: Set $M' \leftarrow \emptyset$, $x_j \leftarrow 0$ for all $j \in N$ and $u_i \leftarrow 0$ for all $i \in M$.

Step 2: If $M' = M$ holds, go to Step 3. Otherwise compute Δ by (6.5) and set $u_i \leftarrow u_i + \Delta$ for all $i \in M \setminus M'$. Set $x_j \leftarrow 1$ and $M' \leftarrow M' \cup S_j$ for all columns $j \in N$ satisfying $\sum_{i \in M} a_{ij} u_i = c_j$, and then return to Step 2.

Step 3: If there exists a redundant column $j \in N$, set $x_j \leftarrow 0$ and return to Step 3; otherwise output $\boldsymbol{x}$ and halt.

Since the improved primal-dual algorithm selects columns $j \in N$ with the minimum score $\tilde{c}_j(\boldsymbol{u})/|S_j \setminus M'|$ at each iteration, we can regard the primal-dual algorithm as a variant of the greedy algorithm. Although the running time of a naive implementation of the primal-dual algorithm is $O(mn)$, it can be improved in the same way as the greedy algorithm.

There have been a few computational studies of construction algorithms. Grossman and Wool [37] conducted a comparative study of nine construction algorithms for unweighted SCP (i.e., all columns have a uniform cost). Gomes et al. [36] investigated empirical performance of seven construction algorithms for VCP and SCP.

We now compare five construction algorithms: (i) the greedy algorithm (denoted GR), (ii) the basic primal-dual algorithm (denoted BPD), (iii) the primal-dual algorithm with uniform increase rule (denoted PD-UI), (iv) the primal-dual algorithm using a dual optimal solution $\bar{\boldsymbol{u}}$ of the LP relaxation problem (denoted PD-LP) and (v) the rounding algorithm (denoted RND). We note that PD-LP and RND use primal and dual optimal solutions $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{u}})$ of the LP relaxation problem obtained by CPLEX 9.1.3. We also note that GR, BPD and PD-UI remove redundant columns by the reverse delete step and PD-LP and RND remove redundant columns in the ascending order of values of the optimal solution $\bar{\boldsymbol{x}}$ of the LP relaxation problem. Tables 8 and 9 show upper bounds and computation time in seconds respectively, where computation time of PD-LP and RND does not include time for solving the LP relaxation problem.

From Tables 8 and 9, we can observe that BPD is quite fast but much worse than the other algorithms. This is because the dual feasible solution $\boldsymbol{u}$ obtained by BPD is much

Table 8: Computational results of the construction algorithms

Instance	Rows	Columns	GR	BPD	PD-UI	PD-LP	RND
4.1–4.10	200	1000	528.3	602.2	534.0	521.0	518.6
5.1–5.10	200	2000	270.4	292.5	271.2	277.6	265.7
6.1–6.5	200	1000	156.4	176.4	155.0	161.2	156.6
A.1–A.5	300	3000	253.2	302.8	254.4	255.6	261.0
B.1–B.5	300	3000	78.2	89.4	82.2	85.0	84.6
C.1–C.5	400	4000	234.8	278.4	238.0	244.4	249.8
D.1–D.5	400	4000	70.4	71.8	69.6	71.4	72.8
E.1–E.5	500	5000	31.0	38.4	32.0	31.8	34.8
F.1–F.5	500	5000	16.2	18.8	17.2	17.0	17.4
G.1–G.5	1000	10,000	178.8	212.0	177.4	187.2	192.6
H.1–H.5	1000	10,000	66.6	82.8	66.8	69.0	73.0
RAIL507	507	63,009	210	355	222	201	198
RAIL516	516	47,311	202	368	230	185	185
RAIL582	582	55,515	247	432	285	247	225
RAIL2536	2536	1,081,841	882	1412	987	750	748
RAIL2586	2586	920,683	1157	1755	1290	1057	1057
RAIL4284	4284	1,092,610	1358	2095	1453	1185	1186
RAIL4872	4872	968,672	1868	2853	2015	1685	1685

worse than that obtained by solving the LP relaxation problem such as PD-LP and RND. We also observe that GR is most effective for its computational effort because PD-LP and RND need much computational effort for solving the LP relaxation problem. On the other hand, PD-LP and RND give better upper bounds than those of GR for RAIL instances. This is because many candidates have the same cost effectiveness for RAIL instances.

6.2. Lagrangian heuristics

Solutions of the Lagrangian relaxation problem can be used to construct feasible solutions to the original SCP, which is called the *Lagrangian heuristics*. The Lagrangian heuristics starts with an optimal solution of the Lagrangian relaxation problem $\tilde{\boldsymbol{x}}(\boldsymbol{u})$ and tries to convert it into a feasible solution $\boldsymbol{x}$ to the original SCP, where the greedy algorithm and the primal-dual algorithm are often used for this purpose.

Beasley [8] and Haddadi [38] proposed a Lagrangian heuristic algorithm, which generates a feasible solution for the original SCP at every iteration of the subgradient method. We describe the basic procedure to generate a feasible solution $\boldsymbol{x} = (x_1, \dots, x_n)$ for the original SCP from an optimal solution $\tilde{\boldsymbol{x}}(\boldsymbol{u}) = (\tilde{x}_1(\boldsymbol{u}), \dots, \tilde{x}_n(\boldsymbol{u}))$ of the Lagrangian relaxation problem.

```
Basic Lagrangian heuristic algorithm
Step 1: Set $x_j \leftarrow \tilde{x}_j(\boldsymbol{u})$ for all $j \in N$ and $M' \leftarrow \bigcup_{j \in N,\, \tilde{x}_j(\boldsymbol{u})=1} S_j$.
Step 2: If $M' = M$ holds, go to Step 3. Otherwise, select an uncovered row $i \in$
        $M \setminus M'$, set $x_j \leftarrow 1$ and $M' \leftarrow M' \cup S_j$ for the column $j \in N_i$ with the minimum
        cost $c_j$ and return to Step 2.
Step 3: If there exists a redundant column $j \in N$, set $x_j \leftarrow 0$ and return to Step 3;
        otherwise output $\boldsymbol{x}$ and halt.
```

A number of computational studies have shown that almost equivalent near optimal Lagrangian multiplier vectors can produce upper bounds of substantially different quality. It is therefore worthwhile applying the Lagrangian heuristics for several near optimal Lagrangian

Table 9: Computation time in seconds of the construction algorithms

Instance	Rows	Columns	GR	BPD	PD-UI	PD-LP†	RND†
4.1–4.10	200	1000	<0.01	<0.01	<0.01	<0.01	<0.01
5.1–5.10	200	2000	<0.01	<0.01	<0.01	<0.01	<0.01
6.1–6.5	200	1000	<0.01	<0.01	<0.01	<0.01	<0.01
A.1–A.5	300	3000	<0.01	<0.01	<0.01	<0.01	<0.01
B.1–B.5	300	3000	0.01	<0.01	<0.01	<0.01	<0.01
C.1–C.5	400	4000	0.01	<0.01	<0.01	<0.01	<0.01
D.1–D.5	400	4000	<0.01	<0.01	<0.01	<0.01	<0.01
E.1–E.5	500	5000	0.01	<0.01	<0.01	<0.01	<0.01
F.1–F.5	500	5000	0.01	<0.01	0.01	<0.01	<0.01
G.1–G.5	1000	10,000	<0.01	<0.01	0.01	<0.01	<0.01
H.1–H.5	1000	10,000	0.01	<0.01	0.01	<0.01	<0.01
RAIL507	507	63,009	0.02	<0.01	0.02	0.02	<0.01
RAIL516	516	47,311	0.02	<0.01	0.02	0.02	<0.01
RAIL582	582	55,515	0.02	<0.01	0.02	0.02	<0.01
RAIL2536	2536	1,081,841	0.92	0.09	0.88	0.36	0.34
RAIL2586	2586	920,683	0.83	0.08	0.63	0.30	0.27
RAIL4284	4284	1,092,610	1.11	0.08	1.00	0.38	0.38
RAIL4872	4872	968,672	0.83	0.09	0.67	0.34	0.30

†Note: The computation time of PD-LP and RND does not include the time for solving the LP relaxation problem.

multiplier vectors. As in the case of the greedy algorithm, the Lagrangian heuristics also has many criteria to select the next column; e.g., the reduced cost $\tilde{c}_j(\boldsymbol{u})$ is often used instead of the original cost c_j. This is based on the observation that the reduced cost $\tilde{c}_j(\boldsymbol{u})$ gives a reliable information on the attractiveness of letting $x_j \leftarrow 1$, because each column $j \in N$ with $x_j^* = 1$ in an optimal solution $\boldsymbol{x}^*$ of the original SCP tends to have a small reduced cost $\tilde{c}_j(\boldsymbol{u})$.

Balas and Carrera [3] proposed another Lagrangian heuristic algorithm, which first transforms the Lagrangian multiplier vector $\boldsymbol{u}$ into a dual feasible solution of the LP relaxation problem and then applies the basic primal-dual algorithm.

Tables 10 and 11 show upper bounds and computation time in seconds of four variants of Lagrangian heuristic algorithms: (i) the greedy algorithm using original cost c_j (denoted LH-OC), (ii) the greedy algorithm using reduced cost $\tilde{c}_j(\boldsymbol{u})$ (denoted LH-RC), (iii) the greedy algorithm using cost effectiveness $c_j/|S_j \setminus M'|$ (denoted LH-CE) and (iv) the basic primal-dual algorithm (denoted LH-PD), where each Lagrangian heuristic algorithm is applied at every iteration and removes redundant columns by the reverse delete step. From Tables 10 and 11, we observe that LH-RC, LH-CE and LH-PD are more promising than LH-OC for RAIL instances, while only LH-CE is more promising than LH-OC for classes 4–6 and A–H, although computing the cost effectiveness is expensive. We note that these Lagrangian heuristic algorithms are not necessarily applied at every iteration in the subgradient method since it is quite time consuming for huge SCP instances.

We also note that the Lagrangian heuristics is helpful to fix variables x_j to zero or one because it often improves the best upper bound z_{UB}, and our computational results show that we can increase the number of fixed variables by 9.5% on the average using the Lagrangian heuristics.

Table 10: Computational results of the Lagrangian heuristic algorithms

Instance	Rows	Columns	LH-OC	LH-RC	LH-CE	LH-PD
4.1–4.10	200	1000	510.2	510.2	510.2	510.5
5.1–5.10	200	2000	257.3	257.6	257.4	258.2
6.1–6.5	200	1000	145.8	145.6	145.0	148.0
A.1–A.5	300	3000	243.2	243.0	243.0	245.4
B.1–B.5	300	3000	75.6	76.0	75.6	77.0
C.1–C.5	400	4000	227.6	227.6	227.0	230.0
D.1–D.5	400	4000	65.0	65.2	64.8	66.4
E.1–E.5	500	5000	29.4	29.4	29.0	29.6
F.1–F.5	500	5000	15.0	15.0	14.2	15.0
G.1–G.5	1000	10,000	174.8	175.0	173.8	178.4
H.1–H.5	1000	10,000	63.6	63.8	62.6	64.8
RAIL507	507	63,009	198	191	191	188
RAIL516	516	47,311	202	194	193	197
RAIL582	582	55,515	236	222	226	221
RAIL2536	2536	1,081,841	838	768	791	766
RAIL2586	2586	920,683	1117	1047	1080	1041
RAIL4284	4284	1,092,610	1243	1167	1218	1169
RAIL4872	4872	968,672	1779	1672	1719	1673

6.3. The state-of-the-art heuristic algorithms

In this section, we review the most effective heuristic algorithms for SCP. Many of them are based on the Lagrangian relaxation. Since a better Lagrangian multiplier vector $\boldsymbol{u}$ does not necessarily derive a better upper bound $z(\boldsymbol{x})$, many heuristic algorithms explore both good Lagrangian multiplier vectors $\boldsymbol{u}$ and feasible solutions $\boldsymbol{x}$ of the original SCP simultaneously.

Ceria et al. [24] proposed a Lagrangian heuristic algorithm based on the primal-dual subgradient method as described in Section 3. To deal with huge SCP instances, their algorithm first defines a good core problem $C \subset N$ based on a near optimal Lagrangian multiplier vector $\boldsymbol{u}$ obtained by the primal-dual subgradient method. Differently from the sifting method in Section 4, their algorithm determines the core problem C in a careful way and never changes it afterwards. Their Lagrangian heuristic algorithm applies the primal-dual subgradient method, fixes a variable x_j to one which is selected by its reduced cost $\tilde{c}_j(\boldsymbol{u})$ and primal Lagrangian multiplier, and applies a greedy algorithm for remaining columns to generate a feasible solution $\boldsymbol{x}$ of the original SCP. This procedure is repeated with different parameters for the variable selection and the greedy algorithm.

Caprara et al. [20] proposed a three phase heuristic algorithm. The first one is called the subgradient phase shown in Section 3. The second one is called the heuristic phase, which generates a sequence of near optimal Lagrangian multiplier vectors $\boldsymbol{u}$ and applies a greedy algorithm using reduced costs $\tilde{c}_j(\boldsymbol{u})$. The third one is called the column fixing phase, which fixes the first k columns selected by the greedy algorithm in order to reduce the size of rows and columns to be explored by the three phase heuristic algorithm. The above three phases are repeated until the incumbent solution $\boldsymbol{x}$ cannot be improved further. The three phase heuristic algorithm works on a core problem $C \subset N$ defined by a small subset of columns, which is periodically redefined as described in Section 4.

Wedelin [49] proposed another type of Lagrangian heuristics for a special type of 0-1 integer programming problem, which is a generalization of the set covering and partitioning problems. Differently from the subgradient method, he used the *coordinate ascent method* (or *nonlinear Gauss-Seidel method*) [13] to solve the Lagrangian dual problem, which iteratively

Table 11: Computation time of the Lagrangian heuristic algorithms

Instance	Rows	Columns	LH-OC	LH-RC	LH-CE	LH-PD
4.1–4.10	200	1000	0.05	0.05	0.13	0.06
5.1–5.10	200	2000	0.09	0.09	0.32	0.10
6.1–6.5	200	1000	0.18	0.18	0.48	0.19
A.1–A.5	300	3000	0.31	0.30	1.25	0.33
B.1–B.5	300	3000	0.69	0.68	1.92	0.69
C.1–C.5	400	4000	0.56	0.53	2.34	0.58
D.1–D.5	400	4000	1.27	1.27	3.54	1.46
E.1–E.5	500	5000	4.78	4.38	9.65	5.67
F.1–F.5	500	5000	9.33	9.49	13.62	13.37
G.1–G.5	1000	10,000	3.64	3.44	16.33	3.87
H.1–H.5	1000	10,000	10.03	10.26	26.48	12.92
RAIL507	507	63,009	8.09	7.00	67.91	11.11
RAIL516	516	47,311	4.91	4.69	53.42	7.24
RAIL582	582	55,515	8.81	7.73	59.33	11.61
RAIL2536	2536	1,081,841	212.81	225.25	5689.81	430.67
RAIL2586	2586	920,683	165.34	144.19	4889.30	296.48
RAIL4284	4284	1,092,610	269.98	292.02	9796.34	535.92
RAIL4872	4872	968,672	229.42	233.86	8656.24	336.61

maximizes the objective value $z_{\rm LR}(\boldsymbol{u})$ with respect to a number of fixed directions called the coordinate vectors. More precisely, his algorithm modifies only one Lagrangian multiplier u_i at each iteration. Let $\tilde{c}_{j_1}(\boldsymbol{u})$ and $\tilde{c}_{j_2}(\boldsymbol{u})$ be the first and second smallest reduced costs $\tilde{c}_j(\boldsymbol{u})$ among all $j \in N_i$. The new multiplier u_i is computed by $u_i \leftarrow u_i + (\tilde{c}_{j_1}(\boldsymbol{u}) + \tilde{c}_{j_2}(\boldsymbol{u}))/2$, which makes exactly one column $j \in N_i$ to have negative reduced cost. It iteratively applies the above operation to obtain sufficiently good lower and upper bounds. Since it is often the case that more than one variables have the smallest reduced cost, he also developed an improved algorithm that distorts the reduced costs to eliminate ties. Byun [19] reported computational results of the coordinate ascent method for the set partitioning problem.

Another approach to obtain good upper bounds is metaheuristics, which combines basic heuristic algorithms such as greedy and local search algorithms with sophisticated strategies. Those metaheuristic algorithms for SCP include probabilistic greedy [28], simulated annealing [18, 44], genetic algorithm [2, 11] and so on. In recent years, metaheuristic algorithms incorporated with mathematical programming techniques have been proposed [22, 50].

Yagiura et al. [50] proposed a local search algorithm with a large neighborhood called the 3-flip neighborhood. As the size of 3-flip neighborhood is $O(n^3)$, the neighborhood search becomes expensive if implemented naively. To overcome this, they proposed an efficient implementation that reduces the number of candidates in the neighborhood without sacrificing the solution quality. They also incorporated a strategic oscillation mechanism, to guide the search between feasible and infeasible regions alternately. In addition, in order to handle huge instances, they introduce a heuristic variable fixing technique based on the Lagrangian relaxation.

Caserta [22] proposed a tabu search algorithm with a strategic oscillation mechanism. The algorithm can be regarded as a variant of primal-dual heuristics. It is based on a procedure that can construct a dual feasible solution $\boldsymbol{u}$ of the LP relaxation problem from any feasible solution $\boldsymbol{x}$ of the original SCP. The algorithm alternately applies the tabu search algorithm and the subgradient method to explore better lower and upper bounds respectively.

Table 12: Computational results of the state-of-the-art heuristic algorithms

Instance	Rows	Columns	z_{LP}	CNS	CFT	YKI	CA	CPLEX
E.1–E.5	500	5000	21.38	–	28.4	28.4	–	28.6
F.1–F.5	500	5000	8.92	–	14.0	14.0	–	14.2
G.1–G.5	1000	10,000	149.48	167.4	166.4	166.4	–	168.6
H.1–H.5	1000	10,000	45.67	60.4	59.6	59.6	–	61.8
RAIL507	507	63,009	172.15	174	175	174	174	175
RAIL516	516	47,311	182.00	182	182	182	182	*182
RAIL582	582	55,515	209.71	211	211	211	211	*211
RAIL2536	2536	1,081,841	688.40	692	691	691	691	*689
RAIL2586	2586	920,683	935.92	951	948	945	948	979
RAIL4284	4284	1,092,610	1054.05	1070	1065	1064	1063	1089
RAIL4872	4872	968,672	1509.64	1534	1534	1528	1532	1570

Table 12 shows upper bounds of the state-of-the-art heuristic algorithms for SCP: (i) the Lagrangian relaxation based heuristic algorithm by Ceria et al. [24] (denoted CNS), (ii) the Lagrangian relaxation based heuristic algorithm by Caprara et al. [20] (dented CFT), (iii) the 3-flip neighborhood local search by Yagiura et al. [50] (denoted by YKI) and (iv) the tabu search with a primal-dual heuristic algorithm by Caserta [22] (denoted CA). We note that all of these results are taken from the literature and these algorithms have been run on different computers. Time limits (or computation time) and computers used for these algorithms are given as follows.

CNS: 1000 seconds for all instances in classes G and H and RAIL507 and 582, and 10,000 seconds for RAIL2586 and 4872 on an IBM RS/6000 375 (32MB memory). 3000 seconds for RAIL516 on a PC486/66 (16MB memory). 10,000 seconds for RAIL2536 and 4284 on an HP735 (125MHz, 256MB memory).

CFT: 5000 seconds for all instances in classes E–H on a DEC station 5000/240. 3000 seconds for RAIL507–582 on a PC486/33 (4MB memory). 10,000 seconds for RAIL2536–4872 on an HP735 (125MHz, 256MB memory).

YKI: 180 seconds for all instances in classes E–H, 600 seconds for RAIL507 and 516, 30 seconds for RAIL582, and 18,000 seconds for RAIL2536–4872 on a Sun Ultra 2 Model 2300 with two Ultra SPARC II processors (300MHz, 1GB memory). We note that we show the best results of ten runs for each instance in Table 12.

CA: 139 seconds for RAIL507, 217 seconds for RAIL516, 131 seconds for RAIL582, 338 seconds for RAIL2536, 399 seconds for RAIL2586, 1022 seconds for RAIL4284 and 1166 seconds for RAIL4872 on a Linux workstation with Intel Pentium 4 (1.1GHz, 256MB memory).

Comparisons of the performance of different computers are found in Dongarra [26] and SPEC (Standard Performance Evaluation Corporation) [53]. In addition, we also show upper bounds obtained by a general purpose MIP (mixed integer programming) solver called CPLEX 9.1.3 [51], where we set the time limit for each run to be 180 seconds for all instances in classes E–H, 600 seconds for RAIL507–582 and 18,000 seconds for RAIL2536–4872, respectively. Optimal values are marked with asterisks.

From Table 12, we observe that CPLEX obtained optimal solutions for RAIL516, 582 and 2536 with 4.38, 63.70 and 9630.66 seconds, respectively. However, the state-of-the-art heuristic algorithms achieve comparable upper bounds for classes E–H and RAIL507–582, and better upper bounds for RAIL2586–4872.

7. Conclusion

In this paper, we reviewed a number of heuristic algorithms to obtain good lower and upper bounds for SCP, including the linear programming, the subgradient method, construction algorithms, metaheuristics and their combinations. In particular, we focus on contributions of mathematical programming techniques, which provide good lower bounds and can also help to obtain good upper bounds when incorporated with heuristic algorithms such as greedy methods, local search and metaheuristic algorithms.

We expect that the hybridization of metaheuristics and mathematical programming approaches will be helpful to handle large-scale instances of other combinatorial optimization problems such as the 0-1 integer programming problem.

The survey in this paper is by no means comprehensive, but we hope this article gives useful information for people who are interested in devising efficient algorithms for this basic problem, which is of practical as well as theoretical importance. We refer the interested reader to introductions to the set covering and related problems by Balas and Padberg [5], Ceria et al. [23], Hochbaum [42] and Caprara et al. [21].

Acknowledgments

The authors are grateful to anonymous referees for their valuable comments, which were used to improve this paper. This research was partially supported by a Scientific Grant-in-Aid from the Ministry of Education, Culture, Sports, Science and Technology of Japan.

References

[1] A. Agrawal, P. Klein, and R. Ravi: When trees collide: An approximation algorithm for the generalized Steiner problem on networks. *SIAM Journal on Computing*, **24** (1995), 440–456.

[2] K.S. Al-Sultan, M.F. Hussain, and J.S. Nizami: A genetic algorithm for the set covering problem. *Journal of the Operational Research Society*, **47** (1996), 702–709.

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

[4] E. Balas and A. Ho: Set covering algorithms using cutting planes, heuristics, and subgradient optimization: A computational study. *Mathematical Programming*, **12** (1980), 37–60.

[5] E. Balas and W. Padberg: Set partitioning: A survey. *SIAM Review*, **18** (1976), 710–761.

[6] R. Bar-Yehuda and S. Even: A linear-time approximation algorithm for the weighted vertex cover problem. *Journal of Algorithms*, **2** (1981), 198–203.

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

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

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

[10] J.E. Beasley: Lagrangean relaxation. In C.R. Reeves (ed.): *Modern Heuristic Techniques for Combinatorial Problems* (Blackwell Scientific Publications, 1993), 243–303.

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

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

[13] D.P. Bertsekas: *Nonlinear Programming* (Athena Scientific, 1995).

[14] R.E. Bixby: Solving real-world linear programs: A decade and more of progress. *Operations Research*, **50** (2002), 3–15.

[15] R.E. Bixby, J.W. Gregory, I.J. Lustig, R.E. Marsten, and D.F. Shanno: Very large-scale linear programming: A case study in combining interior point and simplex methods. *Operations Research*, **40** (1992), 885–897.

[16] J. Borneman, M. Chrobak, G.D. Vedova, A. Figueroa, and T. Jiang: Probe selection algorithms with applications in the analysis of microbial communities. *Bioinformatics*, **17** (2001), S39–S48.

[17] E. Boros, P.L. Hammer, T. Ibaraki, and A. Kogan: Logical analysis of numerical data. *Mathematical Programming*, **79** (1997), 163–190.

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

[19] C.Y. Byun: Lower bounds for large-scale set partitioning problems. *ZIB-Report*, **01–06** (Zuse Institute Berlin, 2001).

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

[21] A. Caprara, P. Toth, and M. Fischetti: Algorithms for the set covering problem. *Annals of Operations Research*, **98** (2000), 353–371.

[22] M. Caserta: Tabu search-based metaheuristic algorithm for large-scale set covering problems. In K.F. Doerner, M. Gendreau, P. Greistorfer, W.J. Gutjahr, R.F. Hartl, and M. Reimann (eds.): *Metaheuristics: Progress in Complex Systems Optimization* (Springer, 2007), 43–63.

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

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

[25] V. Chvátal: A greedy heuristic for the set covering problem. *Mathematics of Operations Research*, **4** (1979), 233–235.

[26] J.J. Dongarra: Performance of various computers using standard linear equations software. Technical Report, **CS89–85** (Computer Science Department, University of Tennessee, 2005).

[27] U. Feige: A threshold of $\ln n$ for approximating set cover. *Journal of the ACM*, **45** (1998), 634–652.

[28] A. Feo and M.G.C. Resende: A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, **8** (1989), 67–71.

[29] M.L. Fisher: The Lagrangian relaxation method for solving integer programming problems. *Management Science*, **27** (1981), 1–18.

[30] M.L. Fisher and P. Kedia: Optimal solution of set covering/partitioning problems using dual heuristics. *Management Science*, **36** (1990), 674–688.

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

[32] R.S. Garfinkel and G.L. Nemhauser: *Integer Programming* (John Wiley & Sons, 1972).

[33] A.M. Geoffrion: Lagrangian relaxation for integer programming. *Mathematical Programming Study*, **2** (1974), 82–114.

[34] M.X. Goemans and D.P. Williamson: A general approximation technique for constrained forest problems. *SIAM Journal on Computing*, **24** (1995), 296–317.

[35] M.X. Goemans and D.P. Williamson: The primal-dual method for approximation algorithms and its application to network design problems. In D.S. Hochbaum (ed.): *Approximation Algorithms for NP-hard Problems* (International Thomson Publishing, 1997), 144–191.

[36] F.C. Gomes, C.N. Meneses, P.M. Pardalos, and G.V. R. Viana: Experimental analysis of approximation algorithms for the vertex cover and set covering problems. *Computers and Operations Research*, **33** (2006), 3520–3534.

[37] T. Grossman and A. Wool: Computational experience with approximation algorithms for the set covering problem. *European Journal of Operational Research*, **101** (1997), 81–92.

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

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

[40] A.C. Ho: Worst case analysis of a class of set covering heuristics. *Mathematical Programming*, **23** (1982), 170–180.

[41] D.S. Hochbaum: Approximation algorithms for the set covering and vertex cover problems. *SIAM Journal on Computing*, **11** (1982), 555–556.

[42] D.S. Hochbaum: Approximating covering and packing problems: set cover, vertex cover, independent set, and related problems. D.S. Hochbaum (ed.): *Approximation Algorithms for NP-hard Problems* (International Thomson Publishing, 1997), 94–143.

[43] K.L. Hoffman and M. Padberg: Solving airline crew scheduling problems by branch-and-cut. *Management Science*, **39** (1993), 657–682.

[44] L.W. Jacobs and M.J. Brusco: Note: A local-search heuristics for large set-covering problems. *Naval Research Logistics*, **42** (1995), 1129–1140.

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

[46] L. Trevisan: Non-approximatability results for optimization problems on bounded degree instances. *Proceedings of the 33rd annual ACM symposium on Theory of computing* (2001), 453–461.

[47] F.J. Vasko and G.R. Wilson: An efficient heuristics for large set covering problems. *Naval Research Logistics Quarterly*, **31** (1984), 163–171.

[48] V.V. Vazirani: *Approximation algorithms* (Springer, 2001).

[49] 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.

[50] M. Yagiura, M. Kishida, and T. Ibaraki: A 3-flip neighborhood local search for the set covering problem. *European Journal of Operational Research*, **172** (2006), 472–499.

[51] ILOG: CPLEX. `http://www.ilog.com/`.

[52] A. Makhorin: GLPK (GNU Linear Programming Kit).
`http://www.gnu.org/software/glpk/`.

[53] SPEC (Standard Performance Evaluation Corporation). `http://www.spec.org/`.

Shunji Umetani
Department of Systems Engineering
Faculty of Electro-Communications
The University of Electro-Communications
1-5-1 Chofugaoka, Chofu
Tokyo 182-8585 Japan
E-mail: `umetani@se.uec.ac.jp`