数理解析研究所講究録
第1829巻 2013年 163-177

一般化上界制約付き集合多重被覆問題に対する発見的解法

梅谷 俊治 (大阪大学), 荒川 正尚 (富士通株式会社), 柳浦睦憲 (名古屋大学)

概 要

集合被覆問題とは，台集合とその部分集合の族および各部分集合のコストが与えられたとき，台集合の全ての要素を被覆するコスト最小の部分集合の組み合わせを求める問題である．大規模な集合被覆問題に対して効率的な発見的解法が提案されている一方で，実際の応用事例では個々の事業者に固有の追加制約をともなうため，現場では整数計画問題に対する汎用ソルバーを利用せざるを得ない場合が少なくない．本研究では，人員スケジューリング問題における職種毎の人数制限，配送計画における車種毎の台数制限，データの論理的解析におけるロバストな解析など応用事例に頻繁に現れる追加制約に対応するため，多重制約と一般化上界制約を集合被覆問題に導入した一般化上界制約付き集合多重被覆問題に対して効率的な発見的解法を提案する．

1 Introduction

The set covering problem (SCP) is one of representative combinatorial optimization problems. We are given a ground set of m elements $i \in M = \{1, \ldots, m\}$, n subsets $S_j \subseteq M(|S_j| \geq 1)$ and costs $c_j(> 0)$ for $j \in N = \{1, \ldots, n\}$. We say that $X \subseteq N$ is a cover of M if $\bigcup_{j \in X} S_j = M$ holds. The goal of SCP is to find a minimum cost cover X of M. The SCP is formulated as a 0-1 integer programming (IP) problem as follows:

$$\begin{array}{lll} \text{(SCP)} \quad \text{minimize} & \displaystyle\sum_{j \in N} c_j x_j & \\ \text{subject to} & \displaystyle\sum_{j \in N} a_{ij} x_j \geq 1, & i \in M, \\ & x_j \in \{0, 1\}, & j \in N, \end{array} \tag{1}$$

where $a_{ij} = 1$ if $i \in S_j$ holds and $a_{ij} = 0$ otherwise, and $x_j = 1$ if $j \in X$ holds and $x_j = 0$ otherwise, respectively. That is, a column $\boldsymbol{a}_j = (a_{1j}, \ldots, a_{mj})^T$ of matrix (a_{ij}) represents the corresponding subset S_j by $S_j = \{i \in M \mid a_{ij} = 1\}$, and the vector $\boldsymbol{x}$ also represents the corresponding cover X by $X = \{j \in N \mid x_j = 1\}$. For notational convenience, for each $i \in M$, let $N_i = \{j \in N \mid a_{ij} = 1\}$ be the index set of subsets S_j that contain element i.

The SCP is known to be NP-hard in the strong sense, and there is no polynomial time approximation scheme (PTAS) unless P = NP. However, the worst-case performance analysis does not necessarily reflect the experimental performance in practice. The continuous development of mathematical programming has much improved the performance of heuristic algorithms accompanied by advances in computational machinery [7, 21]. For example, Beasley [2] presented a number of greedy algorithms based on Lagrangian relaxation (called the Lagrangian heuristics), and Caprara et al. [5] introduced pricing techniques into a Lagrangian heuristic algorithm to reduce the size of instances. Several efficient heuristic algorithms based on Lagrangian heuristics have been developed to solve very large-scale instances with up to 5000 constraints and 1,000,000

variables with deviation within about 1% from the optimum in a reasonable computing time [5, 8, 9, 23].

The SCP is often referred in the literature that it has many important applications, e.g., crew scheduling [5], vehicle routing [17], facility location, and logical analysis of data [4]. However, it is often difficult to formulate problems in real applications into the SCP, because they often have additional side constraints in practice. Most practitioners accordingly formulate them into general mixed integer programming (MIP) problem and apply general purpose solvers, which are usually less efficient compared to solvers specially tailored to SCP.

In this paper, we consider an extension of SCP introducing (i) multicover and (ii) generalized upper bound (GUB) constraints, which arise in many real applications of SCP, e.g., vehicle routing [10], crew scheduling [19], staff scheduling [6, 18] and logical analysis of data [16]. The multicover constraint [20, 22] is a generalization of covering constraint, in which each element $i \in M$ must be covered at least $b_i \in \mathbb{Z}_+$ ($\mathbb{Z}_+$ is the set of non-negative integers) times. GUB constraint is defined as follows. We are given a partition $\{G_1, \ldots, G_k\}$ of N ($\forall h \neq h', G_h \cap G_{h'} = \emptyset, \bigcup_{h=1}^{k} G_h = N$). For each block $G_h \subseteq N$ ($h \in K = \{1, \ldots, k\}$), the number of selected subsets S_j ($j \in G_h$) is constrained to be at most $d_h (\leq |G_h|)$. We call this problem the set multicover problem with GUB constraints (SMCP-GUB).

The SMCP-GUB is NP-hard, and the (supposedly) simpler problem of judging the existence of a feasible solution is NP-complete, since the satisfiability (SAT) problem can be reduced to this problem. In real applications, we often encounter instances with no feasible solution, and it is necessary to find an acceptable solution that violates only a small number of less important constraints. We accordingly consider the following formulation of SMCP-GUB that allows violations of the multicover constraints and introduces a penalty function with a penalty weight vector $\boldsymbol{w} = (w_1, \ldots, w_m) \in \mathbb{R}^m_+$.

$$
\begin{array}{lll}
\text{(SMCP-GUB)} \quad \text{minimize} & z(\boldsymbol{x}) = \displaystyle\sum_{j \in N} c_j x_j + \sum_{i \in M} w_i y_i & \\
\text{subject to} & \displaystyle\sum_{j \in N} a_{ij} x_j + y_i \geq b_i, & i \in M, \\
& \displaystyle\sum_{j \in G_h} x_j \leq d_h, & h \in K, \\
& x_j \in \{0, 1\}, & j \in N, \\
& y_i \in \{0, \ldots, b_i\}, & i \in M.
\end{array} \tag{2}
$$

For a given $\boldsymbol{x} \in \{0,1\}^n$, we can easily compute an optimal $\boldsymbol{y}$ by $y_i = \max\{b_i - \sum_{j \in N} a_{ij} x_j, 0\}$. We note that when $\boldsymbol{y}^* = \boldsymbol{0}$ holds for an optimal solution $(\boldsymbol{x}^*, \boldsymbol{y}^*)$ of SMCP-GUB under the soft multicover constraints, $\boldsymbol{x}^*$ is also optimal under the original (hard) multicover constraints if $w_i > \sum_{j \in N} c_j$ holds for all $i \in M$. In this paper, we accordingly set $w_i = \sum_{j \in N} c_j + 1$ for all $i \in M$.

This generalization of SCP substantially extends the variety of its applications. However, GUB constraints often make the pricing method less effective, because they prevent solutions from containing highly evaluated variables together. To overcome this, we propose a heuristic algorithm to reduce the size of problem instances. In this algorithm, we introduce a new evaluation scheme of variables taking account of GUB constraints. We also develop a 2-flip neighborhood local search algorithm. It features (i) an efficient implementation that reduces

the number of candidates in the neighborhood without sacrificing the solution quality, and (ii) an adaptive control of penalty weights to guide the search to visit better solutions. The latter feature is adopted because when fixed large penalty weights are used, the search tends to stop at locally optimal solutions of low quality. This is because to reach from a good solution to a better one by a sequence of neighborhood operations, it is often unavoidable to temporarily increase the values of some variables y_i, and large penalty weights prevent the algorithm from moving between such solutions. To guide the search to visit a wide variety of good solutions, we also introduce an evolutionary approach called the path relinking method [15] that generates new solutions by combining two or more solutions obtained by then.

Figure 1 illustrates the outline of the entire algorithm for SMCP-GUB. Our algorithm first solves a Lagrangian dual problem to obtain a Lagrangian multiplier vector $\bar{\boldsymbol{u}}$ and a lower bound $z_{\mathrm{LR}}(\bar{\boldsymbol{u}})$ by the subgradient method (Section 2), where it is applied only once in the entire algorithm. Then, our algorithm applies the following three procedures in this order: (i) the heuristic reduction of problem sizes (Section 6), (ii) the path relinking method to generate initial solutions (Section 5), and (iii) the 2-flip neighborhood local search algorithm with the adaptive control of penalty weights (Sections 3 and 4), where they are iteratively applied until a given time limit has run out.

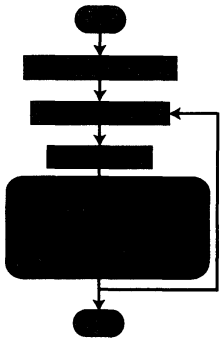

図 1: The outline of the proposed algorithm for SMCP-GUB

2 Lagrangian Relaxation and Subgradient Method

For a given vector $\boldsymbol{u} = (u_1, \ldots, u_m) \in \mathbb{R}^m_+$, called the Lagrangian multiplier vector, the Lagrangian relaxation of SMCP-GUB is defined as follows:

$$
\begin{aligned}
(\mathrm{LR}(\boldsymbol{u})) \quad \text{minimize} \quad z_{\mathrm{LR}}(\boldsymbol{u}) &= \sum_{j\in N} c_j x_j + \sum_{i\in M} w_i y_i + \sum_{i\in M} u_i \left(b_i - \sum_{j\in N} a_{ij} x_j - y_i \right) \\
&= \sum_{j\in N} \left(c_j - \sum_{i\in M} a_{ij} u_j \right) x_j + \sum_{i\in M} y_i (w_i - u_i) + \sum_{i\in M} b_i u_i \qquad (3) \\
\text{subject to} \quad & \sum_{j\in G_h} x_j \le d_h, \qquad h \in K, \\
& x_j \in \{0,1\}, \qquad j \in N, \\
& y_i \in \{0, \ldots, b_i\}, \qquad i \in M,
\end{aligned}
$$

where we call $\tilde{c}_j(\boldsymbol{u}) = c_j - \sum_{i \in M} a_{ij} u_i$ the Lagrangian cost associated with column $j \in N$.

For any $\boldsymbol{u}$, $z_{\mathrm{LR}}(\boldsymbol{u})$ gives a lower bound on the optimal value of SMCP-GUB $z(\boldsymbol{x}^*)$. The problem of finding a Lagrangian multiplier vector $\boldsymbol{u}$ that maximizes $z_{\mathrm{LR}}(\boldsymbol{u})$ is called the Lagrangian dual problem:

$$\text{(LRD)} \quad \text{maximize} \left\{ z_{\mathrm{LR}}(\boldsymbol{u}) \mid \boldsymbol{u} \in \mathbb{R}_+^m \right\}. \tag{4}$$

For a given $\boldsymbol{u}$, we can easily compute an optimal solution to LR($\boldsymbol{u}$). Let $\tilde{\boldsymbol{x}}(\boldsymbol{u}) = (\tilde{x}_1(\boldsymbol{u}), \ldots, \tilde{x}_n(\boldsymbol{u}))$ and $\tilde{\boldsymbol{y}}(\boldsymbol{u}) = (\tilde{y}_1(\boldsymbol{u}), \ldots, \tilde{y}_m(\boldsymbol{u}))$ be an optimal solution to LR($\boldsymbol{u}$). For each block G_h ($h \in K$), if the number of columns $j \in G_h$ satisfying $\tilde{c}_j(\boldsymbol{u}) < 0$ is equal to d_h or less, then set $\tilde{x}_j(\boldsymbol{u}) \leftarrow 1$ (respectively, $\tilde{x}_j(\boldsymbol{u}) \leftarrow 0$) for variables satisfying $\tilde{c}_j(\boldsymbol{u}) < 0$ (respectively, $\tilde{c}_j(\boldsymbol{u}) \geq 0$); otherwise, set $\tilde{x}_j(\boldsymbol{u}) \leftarrow 1$ for variables with the d_h lowest Lagrangian costs $\tilde{c}_j(\boldsymbol{u})$ and $\tilde{x}_j(\boldsymbol{u}) \leftarrow 0$ for the other variables. For $i \in M$, set $\tilde{y}_i(\boldsymbol{u}) \leftarrow b_i$ (respectively, $\tilde{y}_i(\boldsymbol{u}) \leftarrow 0$) if $u_i \geq w_i$ (respectively, $u_i < w_i$) holds.

The Lagrangian relaxation LR($\boldsymbol{u}$) has integrality property, i.e., an optimal solution to the linear programming (LP) relaxation problem of LR($\boldsymbol{u}$) (i.e., the problem obtained by replacing $x_j \in \{0,1\}$ with $0 \leq x_j \leq 1$ for $j \in N$, and $y_i \in \{0, \ldots, b_i\}$ with $0 \leq y_i \leq b_i$ for $i \in M$, respectively) is also optimal to the original problem LR($\boldsymbol{u}$). In this case, any optimal solution $\boldsymbol{u}^*$ to the dual of the LP relaxation problem of SMCP-GUB is also optimal to the Lagrangian dual problem LRD. Hence, the optimal value of the LP relaxation problem of SMCP-GUB z_{LP} is equal to that of LRD $z_{\mathrm{LR}}(\boldsymbol{u}^*)$.

A common approach to compute a near optimal Lagrangian multiplier vector $\boldsymbol{u}$ is the subgradient method. 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

$$s_i(\boldsymbol{u}) = b_i - \sum_{j \in N} a_{ij} \tilde{x}_j(\boldsymbol{u}) - \tilde{y}_i(\boldsymbol{u}). \tag{5}$$

This method generates a sequence of non-negative 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:

$$u_i^{(l+1)} \leftarrow \max \left\{ u_i^{(l)} + \lambda \frac{z_{\mathrm{UB}} - z_{\mathrm{LR}}(\boldsymbol{u}^{(l)})}{||\boldsymbol{s}(\boldsymbol{u}^{(l)})||^2} s_i(\boldsymbol{u}^{(l)}), 0 \right\}, \quad i \in M, \tag{6}$$

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

When huge instances of SCP are solved, the computing time spent on the subgradient method becomes very large if a naive implementation is used. Caprara et al. [5] developed a variant of pricing method on the subgradient method. They define a dual core problem consisting of a small subset of columns $C_d \subset N$ ($|C_d| \ll |N|$), chosen among those having the lowest Lagrangian costs $\tilde{c}_j(\boldsymbol{u})$ ($j \in C_d$), and iteratively update the dual core problem in a similar fashion to that used for solving large scale LP problems. In order to solve huge instances of SMCP-GUB, we also introduce their pricing method into the basic subgradient method (BSM) described in [21].

3 The 2-flip Neighborhood Local Search Algorithm

The local search (LS) starts from an initial solution $\boldsymbol{x}$ and repeats replacing $\boldsymbol{x}$ with a better solution $\boldsymbol{x}'$ in its neighborhood NB($\boldsymbol{x}$) until no better solution is found in NB($\boldsymbol{x}$). For a positive integer r, the r-flip neighborhood $\mathrm{NB}_r(\boldsymbol{x})$ is defined by $\mathrm{NB}_r(\boldsymbol{x}) = \{\boldsymbol{x}' \in \{0,1\}^n \mid d(\boldsymbol{x}, \boldsymbol{x}') \leq r\}$,

where $d(\boldsymbol{x}, \boldsymbol{x}') = |\{j \in N \mid x_j \neq x'_j\}|$ is the Hamming distance between $\boldsymbol{x}$ and $\boldsymbol{x}'$. In other words, $\mathrm{NB}_r(\boldsymbol{x})$ is the set of solutions obtained from $\boldsymbol{x}$ by flipping at most r variables. In our LS, the r is set to 2. In order to improve efficiency, our LS searches $\mathrm{NB}_1(\boldsymbol{x})$ first, and $\mathrm{NB}_2(\boldsymbol{x}) \setminus \mathrm{NB}_1(\boldsymbol{x})$ only if $\boldsymbol{x}$ is locally optimal with respect to $\mathrm{NB}_1(\boldsymbol{x})$.

Since the region searched in a single application of LS is limited, LS is usually applied many times. When a locally optimal solution is obtained, a standard strategy of our algorithm is to update penalty weights and to resume LS from the obtained locally optimal solution. We accordingly evaluate solutions with an alternative evaluation function $\hat{z}(\boldsymbol{x})$, where the original penalty weight vector $\boldsymbol{w}$ is replaced with $\hat{\boldsymbol{w}} = (\hat{w}_1, \ldots, \hat{w}_m) \in \mathbb{R}^m_+$, which are adaptively controlled in the search (See the details in Section 4).

We first describe our LS to search $\mathrm{NB}_1(\boldsymbol{x})$, called the 1-flip neighborhood search. Let

$$\begin{aligned} \Delta\hat{z}^+_j(\boldsymbol{x}) &= c_j - \sum_{i \in M_{\mathrm{L}}(\boldsymbol{x}) \cap S_j} \hat{w}_i, \\ \Delta\hat{z}^-_j(\boldsymbol{x}) &= -c_j + \sum_{i \in (M_{\mathrm{L}}(\boldsymbol{x}) \cup M_{\mathrm{E}}(\boldsymbol{x})) \cap S_j} \hat{w}_i, \end{aligned} \tag{7}$$

denote the increase of $\hat{z}(\boldsymbol{x})$ by flipping $x_j = 0 \to 1$ and $x_j = 1 \to 0$, respectively, where $M_{\mathrm{L}}(\boldsymbol{x}) = \{i \in M \mid \sum_{j \in N} a_{ij} x_j < b_i\}$ and $M_{\mathrm{E}}(\boldsymbol{x}) = \{i \in M \mid \sum_{j \in N} a_{ij} x_j = b_i\}$. Our LS first searches for an improved solution obtainable by flipping $x_j = 0 \to 1$ by searching for $j \in N \setminus X$ satisfying $\Delta\hat{z}^+_j(\boldsymbol{x}) < 0$ and $\sum_{j' \in G_h} x_{j'} < d_h$ for $G_h \ni j$. If an improved solution exist, it chooses j with the minimum $\Delta\hat{z}^+_j(\boldsymbol{x})$; otherwise, it searches for an improved solution obtainable by flipping $x_j = 1 \to 0$ by searching for $j \in X$ satisfying $\Delta\hat{z}^-_j(\boldsymbol{x}) < 0$.

We next describe our LS to search $\mathrm{NB}_2(\boldsymbol{x}) \setminus \mathrm{NB}_1(\boldsymbol{x})$, called the 2-flip neighborhood search. Yagiura et al. [23] developed an LS with the 3-flip neighborhood for SCP. They derived conditions that reduce the number of candidates in $\mathrm{NB}_2(\boldsymbol{x}) \setminus \mathrm{NB}_1(\boldsymbol{x})$ and $\mathrm{NB}_3(\boldsymbol{x}) \setminus \mathrm{NB}_2(\boldsymbol{x})$ without sacrificing the solution quality. However, those conditions are not applicable to the 2-flip neighborhood for SMCP-GUB because of GUB constraints. We therefore propose new conditions that reduce the number of candidates in $\mathrm{NB}_2(\boldsymbol{x}) \setminus \mathrm{NB}_1(\boldsymbol{x})$ taking account of GUB constraints.

Our LS is based on the following three lemmas. Let $\Delta\hat{z}_{j_1,j_2}(\boldsymbol{x})$ denote the increase of $\hat{z}(\boldsymbol{x})$ by flipping the values of x_{j_1} and x_{j_2} simultaneously.

Lemma 1 *Suppose that a solution $\boldsymbol{x}$ is locally optimal with respect to $\mathrm{NB}_1(\boldsymbol{x})$. Then $\Delta\hat{z}_{j_1,j_2}(\boldsymbol{x}) < 0$ holds, only if $x_{j_1} \neq x_{j_2}$.*

The proofs is omitted due to space limitations. Based on this lemma, we consider only the set of solutions obtainable by flipping $x_{j_1} = 1 \to 0$ and $x_{j_2} = 0 \to 1$ simultaneously. We now define

$$\Delta\hat{z}_{j_1,j_2}(\boldsymbol{x}) = \Delta\hat{z}^-_{j_1}(\boldsymbol{x}) + \Delta\hat{z}^+_{j_2}(\boldsymbol{x}) - \sum_{i \in M_{\mathrm{E}}(\boldsymbol{x}) \cap S_{j_1} \cap S_{j_2}} \hat{w}_i. \tag{8}$$

Lemma 2 *Suppose that a solution $\boldsymbol{x}$ is locally optimal with respect to $\mathrm{NB}_1(\boldsymbol{x})$, $x_{j_1} = 1$ and $x_{j_2} = 0$. Then $\Delta\hat{z}_{j_1,j_2}(\boldsymbol{x}) < 0$ holds, only if at least one of the following two conditions holds.*

(i) *Both j_1 and j_2 belong to the same block G_h satisfying $\sum_{j \in G_h} x_j = d_h$.*

(ii) $M_E(\boldsymbol{x}) \cap S_{j_1} \cap S_{j_2} \neq \emptyset$.

Lemma 3 *Suppose that a solution $\boldsymbol{x}$ is locally optimal with respect to* $\mathrm{NB}_1(\boldsymbol{x})$, *and for a block* G_h *and a pair of indices* $j_1, j_2 \in G_h$ *with* $x_{j_1} = 1$ *and* $x_{j_2} = 0$, $\Delta\hat{z}_{j_1,j_2}(\boldsymbol{x}) < 0$ *and* $M_E(\boldsymbol{x}) \cap S_{j_1} \cap S_{j_2} = \emptyset$ *hold. Then we have* $\min_{j \in G_h} \Delta\hat{z}_j^-(\boldsymbol{x}) + \min_{j \in G_h} \Delta\hat{z}_j^+(\boldsymbol{x}) < 0$.

The proof of Lemmas 2 and 3 are omitted due to space limitations. Note that the condition of Lemma 3 implies that the condition (i) of Lemma 2 is satisfied. Then, from Lemma 3, we can conclude that to find an improved solution that satisfies condition (i), it suffices to check only one pair for each block G_h satisfying $\sum_{j \in G_h} x_j = d_h$, instead of checking all pairs (j_1, j_2) with $j_1, j_2 \in G_h$, $x_{j_1} = 1$ and $x_{j_2} = 0$ (provided that the algorithm also checks the solutions that satisfy condition (ii)).

Our LS first searches for an improved solution in $\mathrm{NB}_2(\boldsymbol{x}) \setminus \mathrm{NB}_1(\boldsymbol{x})$ that satisfies the condition (i). For each block G_h $(h \in K)$ that satisfies $\sum_{j \in G_h} x_j = d_h$, it checks the solution obtained by flipping $x_{j_1} = 1 \to 0$ and $x_{j_2} = 0 \to 1$ with the minimum $\Delta\hat{z}_{j_1}^-(\boldsymbol{x})$ and $\Delta\hat{z}_{j_2}^+(\boldsymbol{x})$ $(j_1, j_2 \in G_h)$, respectively. Our LS then searches for an improved solution in $\mathrm{NB}_2(\boldsymbol{x}) \setminus \mathrm{NB}_1(\boldsymbol{x})$ that satisfies the condition (ii). Let $\mathrm{NB}_2^{(j_1)}(\boldsymbol{x})$ denote the subset of $\mathrm{NB}_2(\boldsymbol{x})$ obtainable by flipping $x_{j_1} = 1 \to 0$.

Our LS searches $\mathrm{NB}_2^{(j_1)}(\boldsymbol{x})$ for each $j_1 \in X$ in the ascending order of $\Delta\hat{z}_{j_1}^-(\boldsymbol{x})$. If an improved solution is found, it chooses a pair j_1 and j_2 with the minimum $\Delta\hat{z}_{j_1,j_2}(\boldsymbol{x})$ among those in $\mathrm{NB}_2^{(j_1)}(\boldsymbol{x})$, and it returns to the 1-flip neighborhood search algorithm. Our LS is formally described as follows.

Algorithm LS($\boldsymbol{x}, \hat{\boldsymbol{w}}$)

Input: A solution $\boldsymbol{x}$ and a penalty weight vector $\hat{\boldsymbol{w}}$.

Output: A solution $\boldsymbol{x}$.

Step 1: If $I_1^+(\boldsymbol{x}) = \{j \in N \setminus X \mid \Delta\hat{z}_j^+(\boldsymbol{x}) < 0, \sum_{j' \in G_h} x_{j'} < d_h \text{ for } G_h \ni j\} \neq \emptyset$ holds, choose $j \in I_1^+(\boldsymbol{x})$ with the minimum $\Delta\hat{z}_j^+(\boldsymbol{x})$, set $x_j \leftarrow 1$ and return to Step 1.

Step 2: If $I_1^-(\boldsymbol{x}) = \{j \in X \mid \Delta\hat{z}_j^-(\boldsymbol{x}) < 0\} \neq \emptyset$ holds, choose $j \in I_1^-(\boldsymbol{x})$ with the minimum $\Delta\hat{z}_j^-(\boldsymbol{x})$, set $x_j \leftarrow 0$ and return to Step 2.

Step 3: For each block G_h satisfying $\sum_{j \in G_h} x_j = d_h$ $(h \in K)$, if $\Delta\hat{z}_{j_1,j_2}(\boldsymbol{x}) < 0$ holds for j_1 and j_2 with the minimum $\Delta\hat{z}_{j_1}^-(\boldsymbol{x})$ and $\Delta\hat{z}_{j_2}^+(\boldsymbol{x})$ $(j_1, j_2 \in G_h)$, respectively, set $x_{j_1} \leftarrow 0$ and $x_{j_2} \leftarrow 1$. If the current solution $\boldsymbol{x}$ has been updated at least once in Step 3, return to Step 3.

Step 4: For each $j_1 \in X$ in the ascending order of $\Delta\hat{z}_{j_1}^-(\boldsymbol{x})$, if $I_2(\boldsymbol{x}) = \{j_2 \in N \setminus X \mid \Delta\hat{z}_{j_1,j_2}(\boldsymbol{x}) < 0, \sum_{j' \in G_h} x_{j'} < d_h \text{ for } G_h \ni j_2\} \neq \emptyset$ holds, choose $j_2 \in I_2(\boldsymbol{x})$ with the minimum $\Delta\hat{z}_{j_1,j_2}(\boldsymbol{x})$ and set $x_{j_1} \leftarrow 0$ and $x_{j_2} \leftarrow 1$. If the current solution $\boldsymbol{x}$ has been updated at least once in Step 4, return to Step 1; otherwise output $\boldsymbol{x}$ and exit.

We note that our LS does not necessarily output a locally optimal solution with respect to $\mathrm{NB}_2(\boldsymbol{x})$, because the solution $\boldsymbol{x}$ is not necessarily locally optimal with respect to $\mathrm{NB}_1(\boldsymbol{x})$ in Steps 3 and 4. Though it is easy to keep the solution $\boldsymbol{x}$ locally optimal with respect to $\mathrm{NB}_1(\boldsymbol{x})$ in Steps 3 and 4 by returning to Step 1 whenever an improved solution is obtained in Step 2 or 3, we did not adopt this option because it consumes much computing time just to conclude that the current solution is locally optimal with respect to $\mathrm{NB}_1(\boldsymbol{x})$ in most cases.

Let one-round be the computation needed to find an improved solution in the neighborhood or to conclude that the current solution is locally optimal. For convenience, let $\sigma = \sum_{i \in M} \sum_{j \in N} a_{ij}$, $\tau = \max_{j \in N} \sum_{i \in S_j} |N_i|$, $\nu = \max_{j \in N} |S_j|$ and $n' = \sum_{j \in N} x_j$. If implemented naively, our LS requires $O(\sigma)$ and $O(n\sigma)$ one-round time for $\mathrm{NB}_1(\boldsymbol{x})$ and $\mathrm{NB}_2(\boldsymbol{x})$, respectively. In order to improve efficiency, we make use of the following auxiliary data

$$\begin{aligned} \rho_j^+(\boldsymbol{x}) &= \sum_{i \in M_{\mathrm{L}}(\boldsymbol{x}) \cap S_j} \hat{w}_i, & j \in N \setminus X, \\ \rho_j^-(\boldsymbol{x}) &= \sum_{i \in (M_{\mathrm{L}}(\boldsymbol{x}) \cup M_{\mathrm{E}}(\boldsymbol{x})) \cap S_j} \hat{w}_i, & j \in X. \end{aligned} \tag{9}$$

We store the values of $\rho_j^+(\boldsymbol{x})$ and $\rho_j^-(\boldsymbol{x})$ for $j \in N$ in memory to compute each $\Delta\hat{z}_j^+(\boldsymbol{x}) = c_j - \rho_j^+(\boldsymbol{x})$ and $\Delta\hat{z}_j^-(\boldsymbol{x}) = -c_j + \rho_j^-(\boldsymbol{x})$ in $O(1)$ time. We also store the values of $\theta_i(\boldsymbol{x}) = \sum_{j \in N} a_{ij} x_j$ for $i \in M$ in memory to update the values of $\rho_j^+(\boldsymbol{x})$ and $\rho_j^-(\boldsymbol{x})$ for $j \in N$ in $O(\tau)$ time when $\boldsymbol{x}$ is changed. In our implementation, one-round time is reduced to $O(n + \tau)$ for $\mathrm{NB}_1(\boldsymbol{x})$ and $O(n + k\nu + n'\tau)$ for $\mathrm{NB}_2(\boldsymbol{x}) \setminus \mathrm{NB}_1(\boldsymbol{x})$ by making use of the memory structure. Because $\tau \leq \sigma$ and $n' \leq n$ always hold, these orders are not worse than those of naive implementation, and are much better if $\tau \ll \sigma$ and $n' \ll n$ hold.

4 Adaptive Control of Penalty Weights

Recall that in our algorithm, solutions are evaluated by the alternative evaluation function $\hat{z}(\boldsymbol{x})$ in which the fixed penalty weight vector $\boldsymbol{w}$ in the original objective function $z(\boldsymbol{x})$ is replaced with $\hat{\boldsymbol{w}} = (\hat{w}_1, \ldots, \hat{w}_m) \in \mathbb{R}_+^m$, and the values of $\hat{w}_i$ are adaptively controlled in the search. It is often reported that local search (LS) alone may not attain a sufficiently good solution. Our LS tends to be also attracted to locally optimal solutions of insufficient quality when the original large penalty weights $w_i = \sum_{j \in N} c_j + 1$ $(i \in M)$ are used as $\hat{w}_i$ in the evaluation function $\hat{z}(\boldsymbol{x})$. We accordingly incorporate a mechanism to adaptively control the values of $\hat{w}_i$, i.e., our algorithm iteratively applies LS, updating the penalty weight vector $\hat{\boldsymbol{w}}$ after each call to LS. We call such a sequence of calls to LS *LS-probe*, and use it as the main engine to improve solutions.

Let $\boldsymbol{x}$ denote the solution at which the previous local search stops. The LS-probe resumes LS from $\boldsymbol{x}$ after updating the penalty weight vector $\hat{\boldsymbol{w}}$. Starting from the original penalty weight vector $\hat{\boldsymbol{w}} \leftarrow \boldsymbol{w}$, the penalty weight vector $\hat{\boldsymbol{w}}$ is updated as follows. Let $\boldsymbol{x}^{\mathrm{best}}$ denote the best feasible solution with respect to the original objective function $z(\boldsymbol{x})$ obtained in the current call to LS-probe. If $\hat{z}(\boldsymbol{x}) \geq z(\boldsymbol{x}^{\mathrm{best}})$ holds, LS-probe uniformly decreases the penalty weights $\hat{w}_i \leftarrow (1 - \eta)\hat{w}_i$ for $i \in M$, where the parameter η is adaptively computed so that for 15% of variables satisfying $x_j = 1$, the new value of $\Delta\hat{z}_j^-(\boldsymbol{x})$ becomes negative. Otherwise, LS-probe increases the penalty weights by

$$\hat{w}_i \leftarrow \min\left\{\hat{w}_i \left(1 + \delta \frac{p_i(\boldsymbol{x})}{\max_{i \in M} p_i(\boldsymbol{x})}\right), w_i\right\}, \quad i \in M, \tag{10}$$

where $p_i(\boldsymbol{x}) = \max\{b_i - \sum_{j \in N} a_{ij} x_j, 0\}$ is the amount of violation of the ith multicover constraint, and δ is a parameter that is set to 0.2 in our experiment. LS-probe iteratively applies LS, updating the penalty weight vector $\hat{\boldsymbol{w}}$ after each call to LS, until the best solution obtained in the current call to LS-probe with respect to the original objective function $z(\boldsymbol{x})$ has not improved in the last 50 iterations.

Algorithm LS-probe($\boldsymbol{x}$)

Input: A solution $\boldsymbol{x}$.

Output: The best solution $\boldsymbol{x}^{\text{best}}$ with respect to $z(\boldsymbol{x})$.

Step 1: Set $iter \leftarrow 0$, $\boldsymbol{x}^{\text{best}} \leftarrow \boldsymbol{x}$, $\hat{\boldsymbol{x}} \leftarrow \boldsymbol{x}$ and $\hat{\boldsymbol{w}} \leftarrow \boldsymbol{w}$.

Step 2: Apply LS($\hat{\boldsymbol{x}}, \hat{\boldsymbol{w}}$) to obtain an improved solution $\hat{\boldsymbol{x}}'$. Let $\boldsymbol{x}'$ be the best solution with respect to the original objective function $z(\cdot)$ obtained during the call to LS($\hat{\boldsymbol{x}}, \hat{\boldsymbol{w}}$). Set $\hat{\boldsymbol{x}} \leftarrow \hat{\boldsymbol{x}}'$.

Step 3: If $z(\boldsymbol{x}') < z(\boldsymbol{x}^{\text{best}})$ holds, then set $\boldsymbol{x}^{\text{best}} \leftarrow \boldsymbol{x}'$ and $iter \leftarrow 0$; otherwise set $iter \leftarrow iter + 1$. If $iter \geq 50$ holds, output $\boldsymbol{x}^{\text{best}}$ and halt.

Step 4: If $\hat{z}(\hat{\boldsymbol{x}}) \geq z(\boldsymbol{x}^{\text{best}})$ holds, then uniformly decrease the penalty weights $\hat{w}_i$ for all $i \in M$ by $\hat{w}_i \leftarrow (1 - \eta)\hat{w}_i$; otherwise, increase the penalty weights $\hat{w}_i$ for all $i \in M$ by (10). Return to Step 2.

5 Path Relinking Method

The path relinking [15] is an evolutionary approach to integrate intensification and diversification strategies that generates new solutions by combining two or more solutions. This approach generates new solutions by exploring trajectories that connect good solutions. It starts from one of the good solutions, called an initiating solution, and generates a path by iteratively moving to a solution in the neighborhood that leads toward the other solutions, called guiding solutions.

Because it is preferable to apply path relinking to solutions of high quality, we keep reference sets R_1 and R_2 of good solutions with respect to the original objective function $z(\boldsymbol{x})$ and the alternative evaluation function $\hat{z}(\boldsymbol{x})$ with the current penalty weight vector $\hat{\boldsymbol{w}}$, respectively. Initially R_1 and R_2 are prepared by applying LS-probe to randomly generated solutions. They are then updated by reflecting outcomes of LS-probe whenever LS-probe stops. Suppose that the last call to LS-probe stops at a solution $\boldsymbol{x}$ and $\boldsymbol{x}^{\text{best}}$ is the best solution with respect to $z(\cdot)$ obtained during the last call to LS-probe. Then, the worst solution $\boldsymbol{x}^{\text{worst}}$ in R_1 (with respect to $z(\cdot)$) is replaced with the solution $\boldsymbol{x}^{\text{best}}$ if $z(\boldsymbol{x}^{\text{best}}) \leq z(\boldsymbol{x}^{\text{worst}})$ and $\boldsymbol{x}^{\text{best}} \neq \boldsymbol{x}'$ hold for all $\boldsymbol{x}' \in R_1$. The worst solution $\hat{\boldsymbol{x}}^{\text{worst}}$ in R_2 (with respect to $\hat{z}(\cdot)$) is also replaced with the solution $\boldsymbol{x}$ if $\hat{z}(\boldsymbol{x}) \leq \hat{z}(\hat{\boldsymbol{x}}^{\text{worst}})$ and $\boldsymbol{x} \neq \boldsymbol{x}'$ hold for all $\boldsymbol{x}' \in R_2$.

The path relinking is applied to two solutions $\boldsymbol{x}'$ (initiating solution) and $\boldsymbol{x}''$ (guiding solution) randomly chosen from R_2 and R_1, respectively. Let $\xi = d(\boldsymbol{x}', \boldsymbol{x}'')$ be the Hamming distance between solutions $\boldsymbol{x}'$ and $\boldsymbol{x}''$. It then generates a sequence $\boldsymbol{x}' = \boldsymbol{x}^{(0)}, \boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(\xi)} = \boldsymbol{x}''$ of solutions as follows. Starting from $\boldsymbol{x}^{(0)} \leftarrow \boldsymbol{x}'$, for $l = 1, \ldots, \xi$, it chooses a solution $\boldsymbol{x}^{(l)}$ with the best value of $\hat{z}(\boldsymbol{x})$ among those satisfying $\boldsymbol{x} \in \text{NB}_1(\boldsymbol{x}^{(l-1)})$ and $d(\boldsymbol{x}, \boldsymbol{x}'') < d(\boldsymbol{x}^{(l-1)}, \boldsymbol{x}'')$. Our algorithm chooses the first solution $\boldsymbol{x}^{(l)}$ $(l = 1, \ldots, \xi - 1)$ satisfying $\hat{z}(\boldsymbol{x}^{(l)}) \leq \hat{z}(\boldsymbol{x}^{(l+1)})$ as the next initial solution of LS-probe.

6 Heuristic Reduction of Problem Sizes

For a near optimal Lagrangian multiplier vector $\boldsymbol{u}$, the Lagrangian costs $\tilde{c}_j(\boldsymbol{u})$ give reliable information on the overall utility of selecting columns $j \in N$ for SCP. Based on this property, the

Lagrangian costs $\tilde{c}_j(\boldsymbol{u})$ are often utilized to solve huge instances of SCP, e.g., several heuristic algorithms successively solve a number of subproblems, called primal core problems, consisting of a small subset of columns $C_p \subset N$ ($|C_p| \ll |N|$), chosen among those having low Lagrangian costs $\tilde{c}_j(\boldsymbol{u})$ ($j \in C_p$) [5, 8, 9, 23].

The Lagrangian costs $\tilde{c}_j(\boldsymbol{u})$ are unfortunately unreliable about selecting columns $j \in N$ for SMCP-GUB, because GUB constraints often prevent solutions from containing more than d_h variables x_j with the lowest Lagrangian costs $\tilde{c}_j(\boldsymbol{u})$. To overcome this, we develop an evaluation scheme of columns $j \in N$ for SMCP-GUB taking account of GUB constraints. The main idea of our algorithm is that we modify the Lagrangian costs $\tilde{c}_j(\boldsymbol{u})$ to reduce the number of redundant columns $j \in C_p$ resulting from GUB constraints.

For each block G_h ($h \in K$), let γ_h be the value of the $(d_h + 1)$st lowest Lagrangian cost $\tilde{c}_j(\boldsymbol{u})$ among those for columns in G_h, where we set $\gamma_h \leftarrow 0$ if $d_h = |G_h|$ holds. We then define a score $\hat{c}_j(\boldsymbol{u})$ for a column $j \in G_h$ by $\hat{c}_j(\boldsymbol{u}) = \tilde{c}_j(\boldsymbol{u}) - \gamma_h$ if $\gamma_h < 0$ holds, and $\hat{c}_j(\boldsymbol{u}) = \tilde{c}_j(\boldsymbol{u})$ otherwise. That is, we normalize the Lagrangian costs $\tilde{c}_j(\boldsymbol{u})$ so that at most d_h columns have negative scores $\hat{c}_j(\boldsymbol{u}) < 0$ for each block G_h ($h \in K$). Let $n' = \sum_{j \in N} x_j$ be the number of selected subsets for a solution $\boldsymbol{x}$. Given a solution $\boldsymbol{x}$ and a Lagrangian multiplier vector $\boldsymbol{u}$, a primal core problem is defined by a subset $C_p \subset N$ consisting of (i) columns $j \in N_i$ with the b_i lowest scores $\hat{c}_j(\boldsymbol{u})$ for each $i \in M$, and (ii) columns $j \in N$ with the $10n'$ lowest scores $\hat{c}_j(\boldsymbol{u})$.

Algorithm CORE($\boldsymbol{x}, \boldsymbol{u}$)

Input: A solution $\boldsymbol{x}$ and the Lagrangian multiplier vector $\boldsymbol{u}$.

Output: The primal core problem $C_p \subset N$.

Step 1: For each block G_h ($h \in K$), let γ_h be the value of $(d_h + 1)$st lowest Lagrangian cost $\tilde{c}_j(\boldsymbol{u})$ ($j \in G_h$) if $d_h < |G_h|$ holds and $\gamma_h \leftarrow 0$ otherwise, and then set scores by $\hat{c}_j(\boldsymbol{u}) \leftarrow \tilde{c}_j(\boldsymbol{u}) - \gamma_h$ if $\gamma_h < 0$ holds and $\hat{c}_j(\boldsymbol{u}) \leftarrow \tilde{c}_j(\boldsymbol{u})$ otherwise for all $j \in G_h$.

Step 2: For each $i \in M$, let $C_1(i)$ be the set of columns $j \in N_i$ with the b_i lowest $\hat{c}_j(\boldsymbol{u})$ among those in N_i. Then set $C_1 \leftarrow \bigcup_{i \in M} C_1(i)$.

Step 3: Set C_2 be the set of columns $j \in N$ with the $10n'$ lowest $\hat{c}_j(\boldsymbol{u})$.

Step 4: Set $C_p \leftarrow C_1 \cup C_2$. Output C_p and halt.

The primal core problem C_p is updated before every call to LS-probe. Before updating the primal core problem C_p, our algorithm heuristically fixes some variables x_j to 1 to reflect the characteristics of the incumbent solution $\boldsymbol{x}^*$ and the current solution $\boldsymbol{x}'$. Let $\bar{\boldsymbol{u}}$ be the Lagrangian multiplier vector obtained by the subgradient method, and $V = \{j \in N \mid x^*_j = x'_j = 1\}$ be an index set from which variables to be fixed are chosen. Our algorithm randomly chooses a variable x_j ($j \in V$) with probability

$$\mathrm{prob}_j(\bar{\boldsymbol{u}}) = \frac{\tilde{c}_{\max}(\bar{\boldsymbol{u}}) - \tilde{c}_j(\bar{\boldsymbol{u}})}{\sum_{j' \in V}(\tilde{c}_{\max}(\bar{\boldsymbol{u}}) - \tilde{c}_{j'}(\bar{\boldsymbol{u}}))}, \tag{11}$$

and fixes $x_j = 1$, where $\tilde{c}_{\max}(\bar{\boldsymbol{u}}) = \max_{j' \in V} \tilde{c}_{j'}(\bar{\boldsymbol{u}})$. We note that uniform distribution is used if $\tilde{c}_{\max}(\bar{\boldsymbol{u}}) = \tilde{c}_{j'}(\bar{\boldsymbol{u}})$ holds for all $j' \in V$. Our algorithm iteratively chooses and fixes a variable x_j ($j \in V$) until 20% of multicover constraints are satisfied. It then sets the Lagrangian multiplier

$u_i \leftarrow 0$ if $\sum_{j \in F} a_{ij} \geq b_i$ holds and $u_i \leftarrow \bar{u}_i$ otherwise for $i \in M$, and computes the Lagrangian costs $\tilde{c}_j(\boldsymbol{u})$ for $j \in N \setminus F$, where F is the index set of the fixed variables.

Algorithm FIX($\boldsymbol{x}^*, \boldsymbol{x}', \bar{\boldsymbol{u}}$)

Input: The incumbent solution $\boldsymbol{x}^*$, the current solution $\boldsymbol{x}'$ and the Lagrangian multiplier vector $\bar{\boldsymbol{u}}$.

Output: The set of fixed variables $F \subset N$ and the Lagrangian multiplier vector $\boldsymbol{u}$.

Step 1: Set $V \leftarrow \{j \in N \mid x_j^* = x_j' = 1\}$ and $F \leftarrow \emptyset$.

Step 2: If $\sum_{j \in F} a_{ij} \geq b_i$ holds for 20% of multicover constraints $i \in M$, then for each $i \in M$, set $u_i \leftarrow 0$ if $\sum_{j \in F} a_{ij} \geq b_i$ holds and $u_i \leftarrow \bar{u}_i$ otherwise, output F and $\boldsymbol{u}$, and halt.

Step 3: Randomly choose a column $j \in V$ with probability $\mathrm{prob}_j(\bar{\boldsymbol{u}})$ defined by (11), and set $F \leftarrow F \cup \{j\}$ and $V \leftarrow V \setminus \{j\}$. Return to Step 2.

7 Computational Results

We first prepared eight classes of random instances for SCP, where classes G and H were taken from Beasley's OR Library [3] and classes I–N were newly generated in the same manner, where each class has five instances. We denote instances in class G as G.1, ..., G.5, and other instances in classes H–N similarly. The summary of these instances are given in Table 1, where the density is defined by $\sum_{i \in M} \sum_{j \in N} a_{ij}/mn$ and the costs c_j are random integers taken from interval $[1, 100]$. For each SCP instance, we generate four types of SMCP-GUB instances with different values of parameters d_h and $|G_h|$ as shown in Table 1, where all blocks G_h ($h \in K$) have the same size $|G_h|$ and upper bound d_h for each instance. Here, the right-hand sides of multicover constraints b_i are random integers taken from interval $[1, 5]$.

We compared our algorithm, called the local search algorithm with the heuristic size reduction (LS-SR), with one of the latest mixed integer program (MIP) solver called CPLEX12.3, where they were tested on an IBM-compatible personal computer (Intel Xeon E5420 2.5 GHz, 4 GB memory) and were run on a single thread. Table 1 also shows the time limits in seconds for LS-SR and CPLEX12.3, respectively. We tested two variants of LS-SR: LS-SR1 evaluates variables x_j with the proposed score $\hat{c}_j(\boldsymbol{x})$, and LS-SR2 uses the Lagrangian cost $\tilde{c}_j(\boldsymbol{x})$ in the heuristic reduction of problem sizes. Tables 2–5 show the average objective values (columns "obj.") and the average time to find the best solution (columns "t.t.b.") of LS-SR1, LS-SR2 and CPLEX12.3 for instance types 1–4. The best results among these algorithms are marked with underlines. We also illustrate in Figures 2 and 3 their comparison for each type of SMCP-GUB instances with respect to the relative gap (%)

$$\mathrm{gap}(\boldsymbol{x}) = \frac{z(\boldsymbol{x}) - z_{\mathrm{LP}}}{z_{\mathrm{LP}}} \times 100, \tag{12}$$

where z_{LP} is the optimal value of LP relaxation for SMCP-GUB.

We first observe that LS-SR1 and LS-SR2 achieve better upper bounds than CPLEX12.3 for types 3 and 4 instances. This indicates that LS-SR1 and LS-SR2 are more efficient than CPLEX12.3 for large instances with 10,000 variables or more. One of the main reasons for this is

表 1: The benchmark instances for SMCP-GUB and time limits for our algorithm LS-SR and the MIP solver CPLEX (in seconds)

Instance	Rows	Columns	Density	Instance types ($d_h/\|G_h\|$)				Time limit	
				Type1	Type2	Type3	Type4	LS-SR	CPLEX
G.1–G.5	1000	10,000	2.0%	1/10	10/100	5/10	50/100	600	3600
H.1–H.5	1000	10,000	5.0%	1/10	10/100	5/10	50/100	600	3600
I.1–I.5	1000	50,000	1.0%	1/50	10/500	5/50	50/500	600	3600
J.1–J.5	1000	100,000	1.0%	1/50	10/500	5/50	50/500	600	3600
K.1–K.5	2000	100,000	0.5%	1/50	10/500	5/50	50/500	1200	7200
L.1–L.5	2000	200,000	0.5%	1/50	10/500	5/50	50/500	1200	7200
M.1–M.5	5000	500,000	0.25%	1/50	10/500	5/50	50/500	3000	18,000
N.1–N.5	5000	1,000,000	0.25%	1/100	10/1000	5/100	50/1000	3000	18,000

that the proposed algorithms evaluate a series of candidate solutions efficiently while CPLEX12.3 consumes much computing time for solving LP relaxation problems (even though it uses the warm-start technique for solving a series of LP relaxation problems efficiently). We also observe that LS-SR1 achieves much better upper bounds than those of LS-SR2 and CPLEX12.3 for types 1 and 2 instances. This indicates that LS-SR2 was not able to choose appropriate columns for the primal core problem C_p and LP relaxation based heuristic algorithms in CPLEX12.3 (e.g., local branching [12], feasibility pump [1, 13, 14] and RINS [11]) does not work efficiently for the tested instances. This is probably because the gap between upper and lower bounds is large.

表 2: Computational results of LS-SR and CPLEX12.3 on instance type 1

Instance	z_{LP}	LS-SR1		LS-SR2		CPLEX	
		obj.	t.t.b.	obj.	t.t.b.	obj.	t.t.b.
G.1–G.5	1683.51	2313.4	346.3	2319.8	335.6	2578.0	3286.6
H.1–H.5	395.17	586.6	325.2	589.2	191.5	658.8	1514.9
I.1–I.5	2806.81	3920.4	375.5	5708.4	173.3	4339.0	1882.5
J.1–J.5	1453.48	2012.4	362.8	3977.8	117.2	2361.0	2054.2
K.1–K.5	5593.62	7974.4	951.4	11721.8	64.9	18842.2	515.9
L.1–L.5	2916.64	4178.8	1037.0	8633.6	186.3	5447.2	5295.2
M.1–M.5	5451.55	8424.4	1432.0	18250.8	858.4	19066.0	1555.2
N.1–N.5	4761.81	7951.6	1115.9	20746.2	747.7	18790.0	4841.5

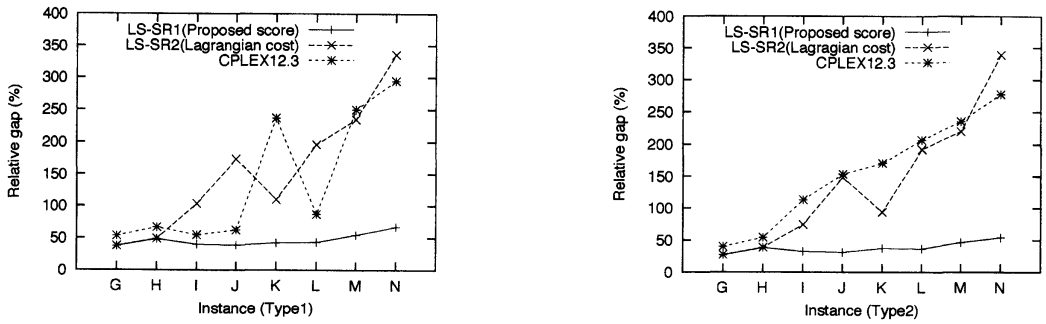

図 2: Comparison of LS-SR and CPLEX12.3 on instance types 1 and 2

表 3: Computational results of LS-SR and CPLEX12.3 on instance type 2

		LS-SR1		LS-SR2		CPLEX	
Instance	z_{LP}	obj.	t.t.b.	obj.	t.t.b.	obj.	t.t.b.
G.1–G.5	1491.11	1888.4	347.2	1896.2	298.6	2084.2	1654.7
H.1–H.5	370.59	512.0	286.0	513.4	271.8	571.6	1227.6
I.1–I.5	2661.28	3518.6	312.6	4637.6	156.2	5663.4	3453.8
J.1–J.5	1382.59	1810.2	389.1	3427.0	326.8	3499.4	3384.0
K.1–K.5	5322.89	7301.2	736.2	10300.4	128.9	14407.2	6608.2
L.1–L.5	2771.20	3780.0	940.6	8089.4	67.5	8491.6	531.7
M.1–M.5	5219.12	7642.0	2060.5	16695.4	1007.3	17504.2	1354.4
N.1–N.5	4596.99	7078.0	1780.3	20177.0	1255.6	17348.4	3277.6

表 4: Computational results of LS-SR and CPLEX12.3 on instance type 3

		LS-SR1		LS-SR2		CPLEX	
Instance	z_{LP}	obj.	t.t.b.	obj.	t.t.b.	obj.	t.t.b.
G.1–G.5	711.02	765.6	261.8	765.6	262.2	830.0	3308.1
H.1–H.5	182.71	205.0	238.8	205.0	239.3	210.0	1186.7
I.1–I.5	930.40	1108.0	243.5	1108.4	300.6	1245.0	2440.9
J.1–J.5	547.47	638.4	423.5	638.0	351.0	693.6	3261.0
K.1–K.5	1851.0	2218.4	846.3	2233.0	633.5	3461.8	6724.1
L.1–L.5	1087.2	1286.8	845.5	1293.0	673.5	2022.2	5298.4
M.1–M.5	2083.5	2575.2	2429.4	2583.2	2534.5	4292.8	1178.9
N.1–N.5	1743.5	2324.4	2282.3	2394.4	1446.1	4444.0	3922.4

We finally compared LS-SR1 with the 3-flip neighborhood local search algorithm proposed by Yagiura et al. [23] (denoted as "YKI") and CPLEX12.3 on the standard SCP instances, where YKI was run on the same conditions as LS-SR1. Table 6 shows the average objective values and the average time to best solution of LS-SR1, YKI and CPLEX12.3 for the standard SCP instances. Figure 4 illustrates their comparisons for the SCP instances with respect to the relative gap (%).

We observe that LS-SR1 achieve comparable upper bounds to those of YKI and better upper bounds than CPLEX12.3, and the gap between upper and lower bounds is still large regardless of multicover and GUB constraints. We accordingly suppose that LS-SR1 achieved good performance for both SMCP-GUB and SCP instances.

8 Conclusion

In this paper, we considered an extension of SCP called the set multicover problem with the generalized upper bound constraints (SMCP-GUB). We proposed a heuristic algorithm to reduce the size of problem instances. For this algorithm, we introduced a new evaluation scheme of variables taking account of GUB constraints. We also developed an efficient implementation of a 2-flip neighborhood local search algorithm. The algorithm reduces the number of candidates in the neighborhood without sacrificing the solution quality. According to computational comparison on benchmark instances with the latest version of a MIP solver called CPLEX12.3, our

表 5: Computational results of LS-SR and CPLEX12.3 on instance type 4

Instance	z_{LP}	LS-SR1		LS-SR2		CPLEX	
		obj.	t.t.b.	obj.	t.t.b.	obj.	t.t.b.
G.1–G.5	690.00	727.0	351.6	727.0	352.3	728.6	2715.2
H.1–H.5	179.40	197.2	208.0	197.2	208.5	200.8	1691.7
I.1–I.5	915.54	1063.8	444.3	1065.0	553.9	1132.6	2893.9
J.1–J.5	537.95	612.0	327.6	611.4	296.4	632.8	2779.1
K.1–K.5	1819.01	2132.6	665.8	2130.6	831.4	3138.0	7052.2
L.1–L.5	1017.17	1235.2	840.0	1234.8	736.9	1932.4	6799.9
M.1–M.5	2052.04	2491.4	2423.0	2492.4	1863.4	3985.2	7097.9
N.1–N.5	1720.12	2230.2	2309.1	2238.2	1638.5	4336.2	2887.7

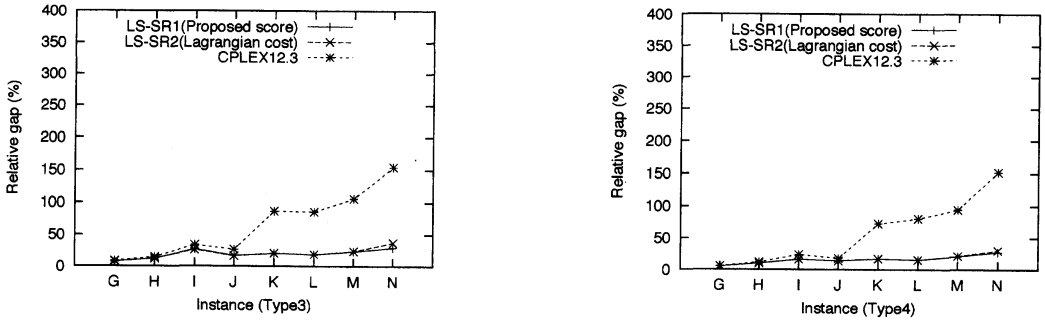

図 3: Comparison of LS-SR and CPLEX12.3 on instance types 3 and 4

algorithm performs quite effectively for various types of instances, especially for very large-scale instances.

参考文献

[1] Achterberg, T., Berthold, T.: Improving the feasible pump. *Discrete Optim.*, **4** (2007), 77–86.

[2] Beasley, J.E.: A Lagrangian heuristic for set-covering problems. *Nav. Res. Logist.*, **37** (1990), 151–164.

[3] Beasley, J.E.: OR-Library: Distributing test problems by electronic mail. *J. Oper. Res. Soc.* **41** (1990), 1069–1072.

[4] Boros, E., Hammer, P.L., Ibaraki, T., Kogan, A.: An implementation of logical analysis of data. *IEEE Trans. Knowl. Data. Eng.*, **12** (2000), 292–306.

[5] Caprara, A., Fischetti, M., Toth, P: A heuristic method for the set covering problem. *Oper. Res.*, **47** (1999), 730–743.

[6] Caprara, A., Monaci, M., Toth, P.: Models and algorithms for a staff scheduling problem. *Math. Program.*, **98** (2003), 445–476.

表 6: Computational results of LS-SR1, YKI and CPLEX12.3 on the standard SCP

		LS-SR1		YKI		CPLEX	
Instance	z_{LP}	obj.	t.t.b.	obj.	t.t.b.	obj.	t.t.b.
G.1–G.5	149.48	166.4	119.3	166.4	2.8	167.6	605.9
H.1–H.5	45.67	59.6	4.4	59.6	1.4	60.8	649.8
I.1–I.5	138.27	158.4	88.5	157.6	22.4	160.6	2966.0
J.1–J.5	104.78	130.8	235.9	129.4	80.0	136.0	2740.1
K.1–K.5	276.66	319.4	267.3	313.8	175.3	321.6	6430.8
L.1–L.5	209.33	263.6	829.0	258.4	445.8	285.2	4640.6
M.1–M.5	415.77	565.8	1554.1	550.4	1265.3	635.6	8452.0
N.1–N.5	348.79	517.6	2048.3	503.8	1517.7	631.4	4274.3

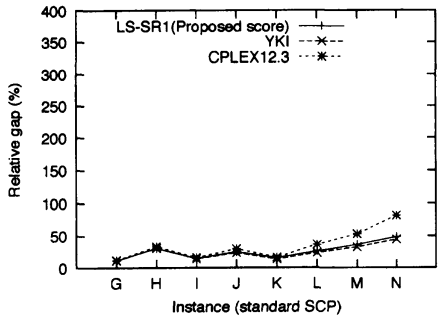

図 4: Comparison of LS-SR1, YKI and CPLEX12.3 on the standard SCP

[7] Caprara, A., Toth, P., Fischetti, M.: Algorithms for the set covering problem. *Ann. Oper. Res.*, **98** (2000), 353–371.

[8] Caserta, M.: Tabu search-based metaheuristic algorithm for large-scale set covering problems. In: Gutjahr, W.J., Hartl, R.F., Reimann, M. (eds.) *Metaheuristics: Progress in Complex Systems Optimization*, pp. 43–63. Springer-Verlag, Berlin, 2007.

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

[10] Choi, E., Tcha, D-W.: A column generation approach to the heterogeneous fleet vehicle routing problem. *Comput. Oper. Res.*, **34** (2007), 2080–2095.

[11] Danna, E., Rothberg, E., Pape, C.L.: Exploring relaxation induced neighborhoods to improve MIP solutions. *Math. Prog.*, **102** (2004), 71–90.

[12] Fischetti, M., Lodi, A.: Local branching. *Math. Prog.*, **98** (2003), 23–47.

[13] Fischetti, M., Glover, F., Lodi, A.: The feasibility pump. *Math. Prog.*, **104** (2005), 91–104.

[14] Fischetti, M., Salvagnin, D.: Feasibility pump 2.0. *Math. Prog. Comp.*, **1** (2009), 201–222.

[15] Glover, F., Laguna, M.: *Tabu Search*, Kluwer Academic Publishers, Massachusetts, 1997.

[16] Hammer, P.L., Bonates, T.O.: Logical analysis of data — An overview: From combinatorial optimization to medical applications. *Ann. Oper. Res.*, **148** (2006), 203–225.

[17] Hashimoto, H., Ezaki, Y., Yagiura, M., Nonobe, K., Ibaraki, T., Løkketangen, A.: A set covering approach for the pickup and delivery problem with general constraints on each route. *Pac. J. Optim.*, **5** (2009), 183–200.

[18] Ikegami, A., Niwa, A.: A subproblem-centric model and approach to the nurse scheduling problem. *Math. Prog.*, **97** (2003), 517–541.

[19] Kohl, N., Karisch, S.E.: Airline crew rostering: Problem types, modeling, and optimization. *Ann. Oper. Res.*, **127** (2004), 223–257.

[20] Pessoa, L.S., Resende, M.G.C., Ribeiro, C.C.: A hybrid Lagrangean heuristic with GRASP and path-relinking for set k-covering. *Comput. Oper. Res.*, in press.

[21] Umetani, S., Yagiura, M.: Relaxation heuristics for the set covering problem. *J. Oper. Res. Soc. Jpn.*, **50** (2007), 350–375.

[22] Vazirani, V.V.: *Approximation algorithms*, Springer-Verlag, Berlin, 2004.

[23] Yagiura, M., Kishida, M., Ibaraki, T.: A 3-flip neighborhood local search for the set covering problem. *Eur. J. Oper. Res.*, **172** (2006), 472–499.