Relaxation heuristics for the set multicover problem with generalized upper bound constraints

Shunji Umetani† Masanao Arakawa‡ Mutsunori Yagiura§

May 16, 2017

Abstract

We consider an extension of the set covering problem (SCP) introducing (i) multicover and (ii) generalized upper bound (GUB) constraints. For the conventional SCP, the pricing method has been introduced to reduce the size of instances, and several efficient heuristic algorithms based on such reduction techniques have been developed to solve large-scale instances. However, GUB constraints often make the pricing method less effective, because they often prevent solutions from containing highly evaluated variables together. To overcome this, we develop heuristic algorithms to reduce the size of instances, in which new evaluation schemes of variables are introduced taking account of GUB constraints. We also develop an efficient implementation of a 2-flip neighborhood local search algorithm that reduces the number of candidates in the neighborhood without sacrificing the solution quality. According to computational comparison on benchmark instances with the recent solvers, the proposed algorithm performs quite effectively for instances having large gaps between lower and upper bounds.

keyword: combinatorial optimization, set covering problem, metaheuristics, local search, Lagrangian relaxation

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 their 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 (0-1 IP) problem as follows:

\[
\begin{align*}
\text{minimize} \quad & \sum_{j \in N} c_j x_j \\
\text{subject to} \quad & \sum_{j \in N} a_{ij} x_j \geq 1, \quad i \in M, \\
& x_j \in \{0, 1\}, \quad j \in N,
\end{align*}
\]

where \( a_{ij} = 1 \) if \( i \in S_j \) holds and \( a_{ij} = 0 \) otherwise, and \( x_j = 1 \) if \( j \in X \) and \( x_j = 0 \) otherwise. That is, a column \( a_j = (a_{1j}, \ldots, a_{mj})^\top \) of matrix \( (a_{ij}) \) represents the corresponding subset \( S_j \) by

\*A preliminary version of this paper was presented in [1].
†Osaka University, Suita, Osaka, 565-0871, Japan. umetani@ist.osaka-u.ac.jp
‡Fujitsu Limited, Kawasaki 211-8588, Japan. arakawa.masanao@jp.fujitsu.com
§Nagoya University, Nagoya 464-8601, Japan. yagiura@nagoya-u.jp
$S_j = \{i \in M \mid a_{ij} = 1\}$. 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 contains the 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 computing machinery [2, 3]. For example, Beasley [4] 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, 6, 7, 8].

The SCP has important real applications such as crew scheduling [5], vehicle routing [9], facility location [10, 11], and logical analysis of data [12]. However, it is often difficult to formulate problems in real applications as SCP, because they often have additional side constraints in practice. Most practitioners accordingly formulate them as general mixed integer programming (MIP) problems 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 such as vehicle routing [13, 14], crew scheduling [15], staff scheduling [16, 17] and logical analysis of data [18]. The multicover constraint is a generalization of covering constraint [19, 20], in which each element $i \in M$ must be covered at least $b_i \in \mathbb{Z}_+$ ($\mathbb{Z}_+$ is the set of nonnegative integers) times. The 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 $S_j$ from the block (i.e., $j \in G_h$) is constrained to be at most $d_h \leq |G_h|$. We call the resulting problem the set multicover problem with GUB constraints (SMCP-GUB), which is formulated as a 0-1 IP problem as follows:

$$\begin{align*}
\text{minimize} & \quad z(x) = \sum_{j \in N} c_j x_j \\
\text{subject to} & \quad \sum_{j \in N} a_{ij} x_j \geq b_i, \quad i \in M, \\
& \quad \sum_{j \in G_h} x_j \leq d_h, \quad h \in K, \\
& \quad x_j \in \{0, 1\}, \quad j \in N.
\end{align*}$$

(2)

This generalization of SCP substantially extends the variety of its applications. However, GUB constraints often make the pricing method less effective, because they often prevent solutions from containing highly evaluated variables together. To overcome this, we develop heuristic algorithms to reduce the size of instances, in which new evaluation schemes of variables are introduced taking account of GUB constraints. We also develop an efficient implementation of a 2-flip neighborhood local search algorithm that reduces the number of candidates in the neighborhood without sacrificing the solution quality. To guide the search to visit a wide variety of good solutions, we also introduce an evolutionary approach called the path relinking method [21] that generates new solutions by combining two or more solutions obtained by them.

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 decision problem. We accordingly allow the search to visit infeasible solutions violating
multicover constraints and evaluate their quality by the following penalized objective function. Note that throughout the remainder of the paper, we do not consider solutions that violate the GUB constraints, and the search only visits solutions that satisfy the GUB constraints. Let $\mathbf{w} = (w_1, \ldots, w_m) \in \mathbb{R}_+^m$ ($\mathbb{R}_+$ is the set of nonnegative real values) be a penalty weight vector. A solution $\mathbf{x}$ is evaluated by

$$
\hat{z}(\mathbf{x}, \mathbf{w}) = \sum_{j \in N} c_j x_j + \sum_{i \in M} w_i \max \left\{ b_i - \sum_{j \in N} a_{ij} x_j, 0 \right\}.
$$

(3)

If the penalty weights $w_i$ are sufficiently large (e.g., $w_i > \sum_{j \in N} c_j$ holds for all $i \in M$), then we can conclude SMCP-GUB to be infeasible when an optimal solution $\mathbf{x}^*$ under the penalized objective function $\hat{z}(\mathbf{x}, \mathbf{w})$ violates at least one multicover constraint. In our algorithm, the initial penalty weights $\bar{w}_i$ ($i \in M$) are set to $\bar{w}_i = \sum_{j \in N} c_j + 1$ for all $i \in M$. Starting from the initial penalty weight vector $\mathbf{w} \leftarrow \bar{w}$, the penalty weight vector $\mathbf{w}$ is adaptively controlled to guide the search to visit better solutions.

We present the outline of the proposed algorithm for SMCP-GUB. The first set of initial solutions are generated by applying a randomized greedy algorithm several times. The algorithm then solves a Lagrangian dual problem to obtain a near optimal Lagrangian multiplier vector $\tilde{\mathbf{u}}$ by a subgradient method (Section 2), which is applied only once in the entire algorithm. Then, the algorithm applies the following procedures in this order: (i) heuristic algorithms to reduce the size of instances (Section 4), (ii) a 2-flip neighborhood local search algorithm (Section 3), (iii) an adaptive control of penalty weights (Section 4), and (iv) a path relinking method to generate initial solutions (Section 5). These procedures are iteratively applied until a given time limit has run out.

## 2 Lagrangian relaxation and subgradient method

For a given vector $\mathbf{u} = (u_1, \ldots, u_m) \in \mathbb{R}_+^m$, called a Lagrangian multiplier vector, we consider the following Lagrangian relaxation problem LR($\mathbf{u}$) of SMCP-GUB:

$$
\text{minimize} \quad z_{LR}(\mathbf{u}) = \sum_{j \in N} c_j x_j + \sum_{i \in M} u_i \left( b_i - \sum_{j \in N} a_{ij} x_j \right) \\
= \sum_{j \in N} \left( c_j - \sum_{i \in M} a_{ij} u_i \right) x_j + \sum_{i \in M} b_i u_i
$$

subject to $\sum_{j \in G_h} x_j \leq d_h$, $h \in K$,

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

We refer to $\tilde{c}_j(\mathbf{u}) = c_j - \sum_{i \in M} a_{ij} u_i$ as the Lagrangian cost associated with column $j \in N$. For any $\mathbf{u} \in \mathbb{R}_+^m$, $z_{LR}(\mathbf{u})$ gives a lower bound on the optimal value $z(\mathbf{x}^*)$ of SMCP-GUB (when it is feasible, i.e., a feasible solution to SMCP-GUB exists).

The problem of finding a Lagrangian multiplier vector $\mathbf{u}$ that maximizes $z_{LR}(\mathbf{u})$ is called the Lagrangian dual problem (LRD):

$$
\text{maximize} \quad \left\{ z_{LR}(\mathbf{u}) \mid \mathbf{u} \in \mathbb{R}_+^m \right\}.
$$

(5)

For a given $\mathbf{u} \in \mathbb{R}_+^m$, we can easily compute an optimal solution $\tilde{x}(\mathbf{u}) = (\tilde{x}_1(\mathbf{u}), \ldots, \tilde{x}_n(\mathbf{u}))$ to LR($\mathbf{u}$) as follows. For each block $G_h$ ($h \in K$), if the number of columns $j \in G_h$ satisfying
\(\bar{c}_j(u) < 0\) is equal to \(d_h\) or less, then set \(\tilde{x}_j(u) \leftarrow 1\) for variables satisfying \(\bar{c}_j(u) < 0\) and \(\tilde{x}_j(u) \leftarrow 0\) for the other variables; otherwise, set \(\tilde{x}_j(u) \leftarrow 1\) for variables with the \(d_h\) lowest Lagrangian costs \(\bar{c}_j(u)\) and \(\tilde{x}_j(u) \leftarrow 0\) for the other variables.

The Lagrangian relaxation problem \(LR(u)\) has integrality property. That is, an optimal solution to \(LR(u)\) is also optimal to its linear programming (LP) relaxation problem obtained by replacing \(x_j \in \{0, 1\}\) in (1) with \(0 \leq x_j \leq 1\) for all \(j \in N\). In this case, any optimal solution \(u^*\) to the dual of the LP relaxation problem of SMCP-GUB is also optimal to LRD, and the optimal value \(z_u^\ast\) of the LP relaxation problem of SMCP-GUB is equal to \(z_u(u^*)\).

A common approach to compute a near optimal Lagrangian multiplier vector \(\tilde{u}\) is the subgradient method. It uses the subgradient vector \(\bar{g}(u) = (g_1(u), \ldots, g_m(u)) \in \mathbb{R}^m\), associated with a given \(u \in \mathbb{R}_+^m\), defined by
\[
g_i(u) = b_i - \sum_{j \in N} a_{ij} \tilde{x}_j(u).
\] (6)

This method generates a sequence of nonnegative Lagrangian multiplier vectors \(u^{(0)}, u^{(1)}, \ldots\), where \(u^{(0)}\) is a given initial vector and \(u^{(l+1)}\) is updated from \(u^{(l)}\) by the following formula:
\[
u_i^{(l+1)} \leftarrow \max \left\{ u_i^{(l)} + \frac{\bar{\epsilon}(x^*, \bar{w}) - z_{u^{(l)}}(\bar{w})}{\|\bar{g}(u^{(l)})\|^2} g_i(u^{(l)}), 0 \right\}, \quad i \in M,
\] (7)
where \(x^*\) is the best solution obtained so far under the penalized objective function \(\bar{\epsilon}(x, \bar{w})\) with the initial penalty weight vector \(\bar{w}\), 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 the pricing method on the subgradient method. They define a core problem consisting of a small subset of columns \(C \subset N\) (\(|C| \ll |N|\)), chosen among those having low Lagrangian costs \(\bar{c}_j(u^{(l)})\) \((j \in N)\). Their algorithm iteratively updates the core problem in a similar fashion that is used for solving large-scale LP problems [22]. In order to solve huge instances of SMCP-GUB, we also introduce a pricing method into the basic subgradient method (BSM) described, e.g., in [3].

### 3 2-flip neighborhood local search

The local search (LS) starts from an initial solution \(x\) and repeats replacing \(x\) with a better solution \(x'\) in its neighborhood \(\text{NB}(x)\) until no better solution is found in \(\text{NB}(x)\). For a positive integer \(r\), the \(r\)-flip neighborhood \(\text{NB}_r(x)\) is defined by \(\text{NB}_r(x) = \{x' \in \{0, 1\}^n \mid d(x, x') \leq r\}\), where \(d(x, x') = |\{j \in N \mid x_j \neq x'_j\}|\) is the Hamming distance between \(x\) and \(x'\). In other words, \(\text{NB}_r(x)\) is the set of solutions obtainable from \(x\) by flipping at most \(r\) variables. We first develop a 2-flip neighborhood local search algorithm (2-FNLS) as a basic component of the proposed algorithm. In order to improve efficiency, 2-FNLS searches \(\text{NB}_2(x)\) first, and then \(\text{NB}_2(x) \setminus \text{NB}_1(x)\).

We first describe the algorithm to search \(\text{NB}_1(x)\), called the 1-flip neighborhood search. Let
\[
\Delta \bar{\epsilon}_j^\dagger(x, w) = \bar{c}_j - \sum_{i \in M_L(x) \setminus S_j} w_i,
\]
\[
\Delta \bar{\epsilon}_j^\ddagger(x, w) = -\bar{c}_j + \sum_{i \in (M_L(x) \cup M_E(x)) \setminus S_j} w_i,
\] (8)
denote the increase of \(\bar{\epsilon}(x, w)\) by flipping \(x_j = 0 \rightarrow 1\) and \(x_j = 1 \rightarrow 0\), respectively, where \(M_L(x) = \{i \in M \mid \sum_{j \in N} a_{ij} x_j < b_i\}\) and \(M_E(x) = \{i \in M \mid \sum_{j \in N} a_{ij} x_j = b_i\}\). The algorithm
first searches for an improved solution obtainable by flipping \( x_j = 0 \to 1 \) by searching for \( j \in N \setminus X(\mathbf{x}) \) satisfying \( \Delta \hat{z}_j^\dagger(\mathbf{x}, \mathbf{w}) < 0 \) and \( \sum_{j' \in G_h} x_{j'} < d_h \) for \( G_h \ni j \), where \( X(\mathbf{x}) = \{ j \in N \mid x_j = 1 \} \). If an improved solution exists, it chooses \( j \) with the minimum \( \Delta \hat{z}_j^\dagger(\mathbf{x}, \mathbf{w}) \); otherwise, it searches for an improved solution obtainable by flipping \( x_j = 1 \to 0 \) by searching for \( j \in X(\mathbf{x}) \) satisfying \( \Delta \hat{z}_j^\dagger(\mathbf{x}, \mathbf{w}) < 0 \).

We next describe the algorithm to search \( \text{NB}_2(\mathbf{x}) \setminus \text{NB}_1(\mathbf{x}) \), called 2-flip neighborhood search. Yagiura et al. [8] developed a 3-flip neighborhood local search algorithm for SCP. They derived conditions that reduce the number of candidates in \( \text{NB}_2(\mathbf{x}) \setminus \text{NB}_1(\mathbf{x}) \) and \( \text{NB}_3(\mathbf{x}) \setminus \text{NB}_2(\mathbf{x}) \) without sacrificing the solution quality. However, those conditions are not directly applicable to the 2-flip neighborhood search for SMCP-GUB because of GUB constraints. We therefore derive the following three lemmas that reduce the number of candidates in \( \text{NB}_2(\mathbf{x}) \setminus \text{NB}_1(\mathbf{x}) \) by taking account of GUB constraints.

Let \( \Delta \hat{z}_{j_1,j_2}(\mathbf{x}, \mathbf{w}) \) denote the increase of \( \hat{z}(\mathbf{x}, \mathbf{w}) \) by flipping the values of \( x_{j_1} \) and \( x_{j_2} \) simultaneously.

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

**Proof.** See A

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 assume that \( \sum_{j \in G_h} x_j < d_h \) holds for \( G_h \ni j_1 \) or \( j_1 \) and \( j_2 \) are in the same block \( G_h \), because otherwise the move is infeasible. Let

\[
\Delta \hat{z}_{j_1,j_2}(\mathbf{x}, \mathbf{w}) = \Delta \hat{z}_{j_1}^\dagger(\mathbf{x}, \mathbf{w}) + \Delta \hat{z}_{j_2}^\dagger(\mathbf{x}, \mathbf{w}) - \sum_{i \in M_E(\mathbf{x}) \setminus S_{j_1} \cap S_{j_2}} w_i
\]

denote the increase of \( \hat{z}(\mathbf{x}, \mathbf{w}) \) in this case.

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

1. Both \( j_1 \) and \( j_2 \) belong to the same block \( G_h \) satisfying \( \sum_{j \in G_h} x_j = d_h \).
2. \( M_E(\mathbf{x}) \cap S_{j_1} \cap S_{j_2} = \emptyset \).

**Proof.** See B

**Lemma 3.** Suppose that a solution \( \mathbf{x} \) is locally optimal with respect to \( \text{NB}_1(\mathbf{x}) \), and for a block \( G_h \) and a pair of indices \( j_1,j_2 \in G_h \) with \( j_1 = 1 \) and \( j_2 = 0 \), \( \Delta \hat{z}_{j_1,j_2}(\mathbf{x}, \mathbf{w}) < 0 \), \( M_E(\mathbf{x}) \cap S_{j_1} \cap S_{j_2} = \emptyset \) and \( \sum_{j \in G_h} x_j = d_h \) hold. Let \( j_1^* = \arg\min_{j \in G_h} \Delta \hat{z}_{j}^\dagger(\mathbf{x}, \mathbf{w}) \) and \( j_2^* = \arg\min_{j \in G_h} \Delta \hat{z}_{j}^\dagger(\mathbf{x}, \mathbf{w}) \). Then we have \( \Delta \hat{z}_{j_1^*,j_2^*}(\mathbf{x}, \mathbf{w}) < 0 \).

**Proof.** See C

Note that the condition of Lemma C implies that the condition (i) of Lemma 2 is satisfied. We can conclude that to find an improved solution satisfying the 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 satisfying the condition (ii) of Lemma 2).

The algorithm first searches for an improved solution \( \mathbf{x}' \in \text{NB}_2(\mathbf{x}) \setminus \text{NB}_1(\mathbf{x}) \) satisfying the condition (i). For each block \( G_h \) \((h \in K)\) satisfying \( \sum_{j \in G_h} x_j = d_h \), it checks the solution obtained by flipping \( x_{j_1} = 1 \to 0 \) and \( x_{j_2} = 0 \to 1 \) for \( j_1 \) and \( j_2 \) in \( G_h \) with the minimum
The algorithm then searches for an improved solution \( x' \in \text{NB}_2(x) \setminus \text{NB}_1(x) \) satisfying the condition (ii). Let \( \text{NB}_2^{(j_1)}(x) \) denote the subset of \( \text{NB}_2(x) \) obtainable by flipping \( x_{j_1} = 1 \to 0 \). The algorithm searches \( \text{NB}_2^{(j_1)}(x) \) for each \( j_1 \in X(x) \) in the ascending order of \( \Delta z^j_{j_1}(x, w) \). If an improved solution is found, it chooses a pair \( j_1 \) and \( j_2 \) with the minimum \( \Delta z_{j_1,j_2}(x, w) \) among those in \( \text{NB}_2^{(j_1)}(x) \).

Algorithm 2-FNLS searches \( \text{NB}_1(x) \) first, and then \( \text{NB}_2(x) \setminus \text{NB}_1(x) \). The algorithm is formally described as follows.

**Algorithm 2-FNLS**

**Input:** A solution \( x \) and a penalty weight vector \( w \).

**Output:** A solution \( x \).

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

**Step 2:** If \( I_1^1(x) = \{ j \in X(x) \mid \Delta z^j_{j}(x, w) < 0 \} \neq \emptyset \) holds, choose \( j \in I_1^1(x) \) with the minimum \( \Delta z^j_{j}(x, w) \), set \( x_j \leftarrow 0 \) and return to Step 2.

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

**Step 4:** For each \( j_1 \in X(x) \) in the ascending order of \( \Delta z^j_{j_1}(x, w) \), if \( I_2(x) = \{ j_2 \in N \setminus X(x) \mid \Delta z^j_{j_2}(x, w) < 0, (\sum_{j \in G_h} x_j < d_h \text{ for } G_h \ni j_2) \text{ or } \exists h, j_1, j_2 \in G_h, M_E(x) \cap S_{j_1} \cap S_{j_2} \neq \emptyset \} \neq \emptyset \) holds, choose \( j_2 \in I_2(x) \) with the minimum \( \Delta z_{j_1,j_2}(x, w) \) and set \( x_{j_1} \leftarrow 0 \) and \( x_{j_2} \leftarrow 1 \). If the current solution \( x \) has been updated at least once in Step 4, return to Step 1; otherwise output \( x \) and exit.

We note that 2-FNLS does not necessarily output a locally optimal solution with respect to \( \text{NB}_2(x) \), because the solution \( x \) is not necessarily locally optimal with respect to \( \text{NB}_1(x) \) in Steps 3 and 4. Though it is easy to keep the solution \( x \) locally optimal with respect to \( \text{NB}_1(x) \) in Steps 3 and 4 by returning to Step 1 whenever an improved solution is obtained in Steps 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 \( \text{NB}_1(x) \) 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, including the time to update relevant data structures and/or memory \[23, 24]. If implemented naively, 2-FNLS requires \( O(\sigma) \) and \( O(n\sigma) \) one-round time for \( \text{NB}_1(x) \) and \( \text{NB}_2(x) \setminus \text{NB}_1(x) \), respectively, where \( \sigma = \sum_{i \in M} \sum_{j \in N} a_{ij} \). In order to improve computational efficiency, we keep the following auxiliary data

\[
\Delta p^j_1(x, w) = \sum_{i \in M(x) \cap S_j} w_i, \quad j \in N \setminus X(x),
\]

\[
\Delta p^j_1(x, w) = \sum_{i \in (M(x) \cup M_E(x)) \cap S_j} w_i, \quad j \in X(x),
\]

in memory to compute each \( \Delta z^j_{j}(x, w) = c_j - \Delta p^j_1(x, w) \) and \( \Delta z^j_{j}(x, w) = -c_j + \Delta p^j_1(x, w) \) in \( O(1) \) time. We also keep the values of \( s_i(x) = \sum_{j \in N} a_{ij}x_j \) \( (i \in M) \) in memory to update the values of \( \Delta p^j_1(x, w) \) and \( \Delta p^j_1(x, w) \) for \( j \in N \) in \( O(\tau) \) time when \( x \) is changed, where \( \tau = \max_{j \in N} \sum_{i \in S_j} |N_i| \) (see \[D\]).
We first consider the one-round time for \( NB_1(x) \). In Steps 1 and 2, the algorithm finds \( j \in N \setminus X(x) \) and \( j \in X(x) \) with the minimum \( \Delta \hat{z}_{j_1}^1(x, w) \) and \( \Delta \hat{z}_{j_2}^2(x, w) \) in \( O(n) \) time, respectively, by using the auxiliary data whose update requires \( O(\tau) \) time. Thus, the one-round time is reduced to \( O(n + \tau) \) for \( NB_1(x) \).

We next consider the one-round time for \( NB_2(x) \setminus NB_1(x) \). In Step 3, the algorithm first finds \( j_1 \) and \( j_2 \) with the minimum \( \Delta \hat{z}_{j_1}^1(x, w) \) and \( \Delta \hat{z}_{j_2}^2(x, w) \) (\( j_1, j_2 \in G_\ell \)), respectively, in \( O(|G_\ell|) \) time. The algorithm then evaluates \( \Delta \hat{z}_{j_1,j_2}(x, w) \) in \( O(\nu) \) time by using \([9]\), where \( \nu = \max_{j \in N} |S_j| \). In Step 4, the algorithm first flips \( x_{j_1} \to 1 \) to \( 0 \) and temporarily updates the values of \( s_i(x) (i \in S_{j_1}) \), \( \Delta p_i^1(x, w) \) and \( \Delta p_i^2(x, w) \) (\( l \in N_i, i \in S_{j_1} \)) in \( O(\tau) \) time so that the memory corresponding to these keeps the values of \( s_i(x') \), \( \Delta p_i^1(x', w) \) and \( \Delta p_i^2(x', w) \) for the \( x' \) obtained from \( x \) by flipping \( x_{j_1} \to 1 \). Then, for searching \( NB_2^{(j_1)}(x) \), the algorithm evaluates

\[
\Delta \hat{z}_{j_1,j_2}(x, w) = \Delta \hat{z}_{j_1}^1(x, w) + \Delta \hat{z}_{j_2}^2(x', w)
\]

\[
= \Delta \hat{z}_{j_1}^1(x, w) + c_{j_2} - \Delta p_{j_2}^2(x', w)
\]

(in \( O(1) \) time for each pair of \( j_1 \) and \( j_2 \)) only for each \( j_2 \in N \setminus X(x) \) such that the value of \( \Delta p_{j_2}^1(x, w) \) has been changed to \( \Delta p_{j_2}^1(x', w) \) (\( \neq \Delta p_{j_2}^1(x, w) \)) during the temporary update. Note that the number of such candidates \( j_2 \) that satisfy \( \Delta p_{j_2}^1(x', w) \neq \Delta p_{j_2}^1(x, w) \) is \( O(\tau) \). When an improved solution was not found in \( NB_2^{(j_1)}(x) \), the updated memory values are restored in \( O(\tau) \) time to the original values \( s_i(x), \Delta p_i^1(x, w) \) and \( \Delta p_i^2(x, w) \) before we try another candidate of \( j_1 \). The time to search \( NB_2^{(j_1)}(x) \) for each \( j_1 \in X(x) \) is therefore \( O(\tau) \). Thus, the one-round time is reduced to \( O(n + \kappa \nu + n' \tau) \) for \( NB_2(x) \setminus NB_1(x) \), where \( n' = \sum_{j \in N} x_j = |X(x)| \).

Because \( k \leq n \), \( \nu \leq m \), \( \tau \leq \sigma \), \( n' \leq n \), \( m \leq \sigma \), and \( n \leq \sigma \) always hold, these orders are not worse than those of naive implementation, and they are much better if \( \nu \ll m \), \( \tau \ll \sigma \) and \( n' \ll n \) hold, which are the case for many instances. We also note that the computation time for updating the auxiliary data has little effect on the total computation time of 2-FNLS, because, in most cases, the number of solutions actually visited is much less than that of evaluated neighbor solutions.

### 4 Adaptive control of penalty weights

We observed that 2-FNLS alone tends to be attracted to locally optimal solutions of insufficient quality when the penalty weights \( w_i \) are large. We accordingly incorporate a mechanism to adaptively control the values of \( w_i (i \in M) \) \([8, 25, 26]\); the algorithm iteratively applies 2-FNLS, updating the penalty weight vector \( w \) after each call to 2-FNLS. We call such a sequence of calls to 2-FNLS the *weighting local search* (WLS) according to \([27, 28]\).

Let \( x \) denote the solution at which the previous 2-FNLS stops. Algorithm WLS resumes 2-FNLS from \( x \) after updating the penalty weight vector \( w \). Starting from an initial penalty weight vector \( w \leftarrow \bar{w} \), where we set \( \bar{w}_i = \sum_{j \in N} c_j + 1 \) for all \( i \in M \), the penalty weight vector \( w \) is updated as follows. Let \( x^{\text{best}} \) denote the best solution obtained in the current call to WLS with respect to the penalized objective function \( \hat{z}(x, \bar{w}) \) with the initial penalty weight vector \( \bar{w} \). If \( \hat{z}(x, w) \geq \hat{z}(x^{\text{best}}, \bar{w}) \) holds, WLS uniformly decreases the penalty weights \( w_i \leftarrow (1 - \eta)w_i \) for all \( i \in M \), where the parameter \( \eta \) is decided so that for 15% of variables satisfying \( x_j = 1 \), the new value of \( \Delta \hat{z}_{j}^2(x, w) \) becomes negative. Otherwise, WLS increases the penalty weights by

\[
w_i \leftarrow \min \left\{ w_i \left(1 + \delta \frac{y_i(x)}{\max_{i \in M} y_i(x)} \right), \bar{w}_i \right\}, \quad i \in M,
\]
where $y_i(x) = \max\{b_i - \sum_{j \in N} a_{ij} x_j, 0\}$ is the amount of violation of the $i$th multicolor constraint, and $\delta$ is a parameter that is set to 0.2 in our computational experiments. Algorithm WLS iteratively applies 2-FNLS, updating the penalty weight vector $\mathbf{w}$ after each call to 2-FNLS, until the best solution $\mathbf{x}^{\text{best}}$ with respect to $\hat{z}(\mathbf{x}, \mathbf{w})$ obtained in the current call to WLS has not improved in the last 50 iterations.

**Algorithm WLS($\mathbf{x}$)**

**Input:** A solution $\mathbf{x}$.

**Output:** A solution $\hat{\mathbf{x}}$ and the best solution $\mathbf{x}^{\text{best}}$ with respect to $\hat{z}(\mathbf{x}, \mathbf{w})$ obtained in the current call to WLS.

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

**Step 2:** Apply 2-FNLS($\hat{\mathbf{x}}, \mathbf{w}$) to obtain an improved solution $\hat{\mathbf{x}}'$ and then set $\hat{\mathbf{x}} \leftarrow \hat{\mathbf{x}}'$ and $\mathbf{x}^{\text{best}} \leftarrow \mathbf{x}'$. Let $\mathbf{x}'$ be the best solution with respect to $\hat{z}(\mathbf{x}, \mathbf{w})$ obtained during the call to 2-FNLS($\hat{\mathbf{x}}, \mathbf{w}$).

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

**Step 4:** If $\hat{z}(\hat{\mathbf{x}}, \mathbf{w}) \geq \hat{z}(\mathbf{x}^{\text{best}}, \mathbf{w})$ holds, then uniformly decrease the penalty weights $w_i$ for all $i \in M$ by $w_i \leftarrow (1 - \eta) w_i$; otherwise, increase the penalty weights $w_i$ for all $i \in M$ by $\{12\}$. Return to Step 2.

## 5 Heuristic algorithms to reduce the size of instances

For a near optimal Lagrangian multiplier vector $\mathbf{u}$, the Lagrangian costs $\tilde{c}_j(\mathbf{u})$ give reliable information on the overall utility of selecting columns $j \in N$ for SCP. Based on this property, the Lagrangian costs $\tilde{c}_j(\mathbf{u})$ are often utilized to solve huge instances of SCP. Similar to the pricing method for solving the Lagrangian dual problem, several heuristic algorithms successively solve a number of subproblems, also called core problems, consisting of a small subset of columns $C \subseteq N$ ($|C| \ll |N|$), chosen among those having low Lagrangian costs $\tilde{c}_j(\mathbf{u})$ ($j \in C$) [3, 5, 7, 8]. The Lagrangian costs $\tilde{c}_j(\mathbf{u})$ are unfortunately unreliable for 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 $\tilde{c}_j(\mathbf{u})$. To overcome this, we develop two evaluation schemes of columns $j \in N$ for SMCP-GUB.

Before updating the core problem $C$ for every call to WLS, the algorithm heuristically fixes some variables $\hat{x}_j \leftarrow 1$ to reflect the characteristics of the incumbent solution $\mathbf{x}'$ and the current solution $\hat{\mathbf{x}}$. Let $\mathbf{u}$ be a near optimal Lagrangian multiplier vector, and $V = \{j \in N \mid x^*_j = \hat{x}_j = 1\}$ be an index set from which variables to be fixed are chosen. Let $\tilde{c}_{\text{max}}(\mathbf{u}) = \max_{j \in V} \tilde{c}_j(\mathbf{u})$ be the maximum value of the Lagrangian cost $\tilde{c}_j(\mathbf{u})$ ($j \in V$). The algorithm randomly chooses a variable $x_j$ ($j \in V$) with probability

$$\text{prob}_j(\mathbf{u}) = \frac{\tilde{c}_{\text{max}}(\mathbf{u}) - \tilde{c}_j(\mathbf{u})}{\sum_{l \in V} (\tilde{c}_{\text{max}}(\mathbf{u}) - \tilde{c}_l(\mathbf{u}))} \quad (13)$$

and fixes $\hat{x}_j \leftarrow 1$. We note that the uniform distribution is used if $\tilde{c}_{\text{max}}(\mathbf{u}) = \tilde{c}_j(\mathbf{u})$ holds for all $j \in V$. The algorithm iteratively chooses and fixes a variable $x_j$ ($j \in V$) until $\sum_{j \in N} a_{ij} x_j \geq b_i$ holds for 20% of multicolor constraints $i \in M$. It then updates the Lagrangian multiplier $u_i \leftarrow 0$ if $\sum_{j \in F} a_{ij} \geq b_i$ holds for $i \in M$, and computes the Lagrangian costs $\tilde{c}_j(\mathbf{u})$ for $j \in N \setminus F$, where $F$ is the index set of the fixed variables. The variable fixing procedure is formally described as follows.
Algorithm \text{FIX}(x^*, \hat{x}, \hat{u})

\textbf{Input:} The incumbent solution $x^*$, the current solution $\hat{x}$ and a near optimal Lagrangian multiplier vector $\hat{u}$.

\textbf{Output:} A set of fixed variables $F \subset N$ and a Lagrangian multiplier vector $u$.

\textbf{Step 1:} Set $V \leftarrow \{j \in N \mid x_j^* = \hat{x}_j = 1\}$, $F \leftarrow \emptyset$, and $u \leftarrow \hat{u}$.

\textbf{Step 2:} If $\{(i \in M \mid \sum_{j \in F} a_{ij} \geq b_i)\} \geq 0.2m$ holds, then set $u_i \leftarrow 0$ for each $i \in M$ satisfying $\sum_{j \in F} a_{ij} \geq b_i$, output $F$ and $u$, and halt.

\textbf{Step 3:} Randomly choose a column $j \in V$ with probability $\text{prob}_j(u)$ defined by \eqref{eq:prob}, and set $F \leftarrow F \cup \{j\}$. Return to Step 2.

Subsequent to the variable fixing procedure, the algorithm updates the instance to be considered by setting $z(x) = \sum_{j \in N\setminus F} c_j x_j + \sum_{j \in F} c_j$, $b_i \leftarrow \max\{b_i - \sum_{j \in F} a_{ij}, 0\}$ (i.e., $i \in M$), and $d_h \leftarrow d_h - |G_h \cap F|$ ($h \in K$).

The first evaluation scheme modifies the Lagrangian costs $\tilde{c}_j(u)$ to reduce the number of redundant columns $j \in C$ resulting from GUB constraints. For each block $G_h$ ($h \in K$), let $\theta_h$ be the value of the $(d_h + 1)^{\text{st}}$ lowest Lagrangian cost $\tilde{c}_j(u)$ among those for columns in $G_h$ if $d_h < |G_h|$ holds and $\theta_h \leftarrow 0$ otherwise. We then define a score $\rho_j$ for $j \in G_h$, called the normalized Lagrangian score, by $\rho_j = \tilde{c}_j(u) - \theta_h$ if $\theta_h < 0$ holds, and $\rho_j = \tilde{c}_j(u)$ otherwise.

The second evaluation scheme modifies the Lagrangian costs $\tilde{c}_j(u)$ by replacing the Lagrangian multiplier vector $u$ with the adaptively controlled penalty weight vector $w$. We define another score $\phi_j$ for $j \in N$, called the pseudo-Lagrangian score, by $\phi_j = \tilde{c}_j(w)$. Intuitive meaning of this score is that we consider a column to be promising if it covers many frequently violated constraints in the recent search. The variable fixing procedure for the second evaluation scheme is described in a similar fashion to that of the first evaluation scheme by replacing the Lagrangian multiplier vectors $\hat{u}$ and $u$ with penalty weight vectors $\hat{w}$ and $w$, respectively.

Given a score vector $\rho$ (resp., $\phi$), a core problem is defined by a subset $C \subset N$ consisting of (i) columns $j \in N_i$ with the $b_i$ lowest scores $\rho_j$ (resp., $\phi_j$) for each $i \in M$, (ii) columns $j \in N$ with the $10n'$ lowest scores $\rho_j$ (resp., $\phi_j$) (recall that we define $n' = \sum_{j \in N} x_j$), and (iii) columns $j \in X(x^*) \cup X(\hat{x})$ for the incumbent solution $x^*$ and the current solution $\hat{x}$. The core problem updating procedure is formally described as follows.

Algorithm \text{CORE}(\rho, x^*, \hat{x})

\textbf{Input:} A score vector $\rho$, the incumbent solution $x^*$ and the current solution $\hat{x}$.

\textbf{Output:} The core problem $C \subset N$.

\textbf{Step 1:} For each $i \in M$, let $C_1(i)$ be the set of columns $j \in N_i$ with the $b_i$ lowest $\rho_j$ among those in $N_i$. Then set $C_1 \leftarrow \bigcup_{i \in M} C_1(i)$.

\textbf{Step 2:} Set $C_2$ be the set of columns $j \in N$ with the $10n'$ lowest $\rho_j$.

\textbf{Step 3:} Set $C \leftarrow C_1 \cup C_2 \cup X(x^*) \cup X(\hat{x})$. Output $C$ and halt.

6 Path relinking

The path relinking method [21] is an evolutionary approach to integrate intensification and diversification strategies. 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 method to solutions of high quality, we keep reference sets $R_1$ and $R_2$ of good solutions with respect to the penalized objective functions $\hat{\varphi}(x, w)$ and $\hat{\varphi}(x, \bar{w})$ with the current penalty weight vector $w$ and the initial penalty weight vector $\bar{w}$, respectively. Initially $R_1$ and $R_2$ are prepared by repeatedly applying a randomized greedy algorithm, which is the same as Steps 1 and 2 of 2-FNLS$(0, \bar{w})$ except for randomly choosing $j \in I_k^R(x)$ from those with the five lowest $\Delta \hat{\varphi}_{i,j}^R(x, \bar{w})$ in Step 1 (recall that we define $\bar{w}$ to be the initial penalty weight vector). Suppose that the last call to WLS stops at a solution $\bar{x}$ and $x^{\text{best}}$ is the best solution with respect to $\hat{\varphi}(x, \bar{w})$ obtained during the last call to WLS. Then, the worst solution $\hat{x}^{\text{best}}$ in $R_1$ (with respect to $\hat{\varphi}(x, \bar{w})$) is replaced with the solution $\bar{x}$ if it satisfies $\hat{\varphi}(\bar{x}, w) \leq \hat{\varphi}(\hat{x}^{\text{best}}, w)$ and $\bar{x} \neq x'$ for all $x' \in R_1$. The worst solution $x^{\text{best}}$ in $R_2$ (with respect to $\hat{\varphi}(x, \bar{w})$) is replaced with the solution $x^{\text{best}}$ if it satisfies $\hat{\varphi}(x^{\text{best}}, \bar{w}) \leq \hat{\varphi}(\hat{x}^{\text{best}}, \bar{w})$ and $x^{\text{best}} \neq x'$ for all $x' \in R_2$.

The path relinking method first chooses two solutions $x^{\text{init}}$ (initiating solution) and $x^{\text{enlk}}$ (guiding solution) randomly, one from $R_1$ and another from $R_2$, where we assume that $\hat{\varphi}(x^{\text{init}}, w) \leq \hat{\varphi}(x^{\text{enlk}}, w)$ and $x^{\text{init}} \neq \bar{x}$ hold. Let $\xi = d(x^{\text{init}}, x^{\text{enlk}})$ be the Hamming distance between solutions $x^{\text{init}}$ and $x^{\text{enlk}}$. It then generates a sequence $x^{(0)} = x^{(0)} = x^{\text{init}}$, $\ldots, x^{(\xi)} = x^{\text{enlk}}$ of solutions as follows. Starting from $x^{(0)} \leftarrow x^{\text{init}}$, for $l = 0, 1, \ldots, \xi - 1$, the solution $x^{(l+1)}$ is defined to be a solution $x^{'}$ with the best value of $\hat{\varphi}(x^{'}, w)$ among those satisfying $x' \in \text{NB}_1(x^{(l)})$ and $d(x^{'}, x^{\text{enlk}}) < d(x^{(l)}, x^{\text{enlk}})$. The algorithm chooses the first solution $x^{(l)}$ ($l = 0, 1, \ldots, \xi - 1$) satisfying $\hat{\varphi}(x^{(l)}, w) \leq \hat{\varphi}(x^{(l+1)}, w)$ as the next initial solution of WLS.

Given a pair of solutions $x$ and $x^{\text{best}}$ and the current reference sets $R_1$ and $R_2$, the path relinking method outputs the next initial solution $x$ of WLS and the updated reference sets $R_1$ and $R_2$. The path relinking method is formally described as follows.

**Algorithm PRL($\bar{x}, x^{\text{best}}, R_1, R_2$)**

**Input:** Solutions $\bar{x}$ and $x^{\text{best}}$ and reference sets $R_1$ and $R_2$.

**Output:** The next initial solution $x$ of WLS and the updated reference sets $R_1$ and $R_2$.

**Step 1:** Let $\hat{x}^{\text{best}} = \arg\max_{x' \in R_1} \hat{\varphi}(x', \bar{w})$ be the worst solution in $R_1$. If the solution $\hat{x}$ satisfies $\hat{\varphi}(\hat{x}, w) \leq \hat{\varphi}(\hat{x}^{\text{best}}, w)$ and $\hat{x} \neq x'$ for all $x' \in R_1$, then set $R_1 \leftarrow R_1 \cup \{\hat{x}\} \setminus \{x^{\text{best}}\}$.

**Step 2:** Let $x^{\text{best}} = \arg\max_{x' \in R_2} \hat{\varphi}(x', \bar{w})$ be the worst solution in $R_2$. If the solution $x^{\text{best}}$ satisfies $\hat{\varphi}(x^{\text{best}}, \bar{w}) \leq \hat{\varphi}(x^{\text{best}}, \bar{w})$ and $x^{\text{best}} \neq x'$ for all $x' \in R_2$, then set $R_2 \leftarrow R_2 \cup \{x^{\text{best}}\} \setminus \{x^{\text{best}}\}$.

**Step 3:** Randomly choose two solutions $x^{\text{init}}$ and $x^{\text{enlk}}$, one from $R_1$ and another from $R_2$, where we assume that $\hat{\varphi}(x^{\text{init}}, w) \leq \hat{\varphi}(x^{\text{enlk}}, w)$ and $x^{\text{init}} \neq \bar{x}$ hold. Set $l \leftarrow 0$ and $x^{(l)} \leftarrow x^{\text{init}}$.

**Step 4:** Set $x^{(l+1)} \leftarrow \arg\min_{x' \in \text{NB}_1(x^{(l)})} \{\hat{\varphi}(x', w) \mid x' \neq x^{(l)}, d(x', x^{\text{enlk}}) < d(x^{(l)}, x^{\text{enlk}})\}$. If $\hat{\varphi}(x^{(l)}, w) > \hat{\varphi}(x^{(l+1)}, w)$ holds, set $l \leftarrow l + 1$ and return to Step 4; otherwise set $x \leftarrow x^{(l)}$, output $x$, $R_1$ and $R_2$, and halt.

### 7 Summary of the proposed algorithm

We summarize the outline of the proposed algorithm for SMCP-GUB in Figure 1. The first reference sets $R_1$ and $R_2$ of good solutions are generated by repeating the randomized greedy algorithm (Section 6). The first initial solution $\bar{x}$ is set to $\bar{x} \leftarrow \arg\min_{x \in R_1 \cup R_2} \hat{\varphi}(x, \bar{w})$. When
using the normalized Lagrangian score $\rho_j$, the algorithm obtains a near optimal Lagrangian multiplier vector $\tilde{u}$ by the basic subgradient method BSM accompanied by a pricing method \[3\] (Section 2), where it is applied only once in the entire algorithm.

The algorithm repeatedly applies the following procedures in this order until a given time has run out. The heuristic variable fixing algorithm $\text{FIX}(x^*, \tilde{x}, \tilde{u})$ (Section 5) decides the index set $F$ of variables to be fixed, and it updates the instance to be considered by fixing variables $\tilde{x}_j \leftarrow 1$ ($j \in F$). The heuristic size reduction algorithm $\text{CORE}(\rho, x^*, \tilde{x})$ (Section 3) constructs a core problem $C \subset N$ and fixes variables $\tilde{x}_j \leftarrow 0$ ($j \in N \setminus C$). The weighting local search algorithm $\text{WLS}(\tilde{x})$ explores good solutions $\hat{x}$ and $x_{\text{best}}$ with respect to $\tilde{z}(x, w)$ and $\hat{z}(x, \tilde{w})$, respectively, by repeating the 2-flip neighborhood local search algorithm 2-FNLS($\tilde{x}, w$) (Section 3) while updating the penalty weight vector $w$ adaptively (Section 4), where the initial penalty weights $\tilde{w}_i$ ($i \in M$) are set to $\tilde{w}_i = \sum_{j \in N} c_{ij} + 1$ for all $i \in M$. After updating the reference sets $R_1$ and $R_2$, the next initial solution $\tilde{x}$ is generated by the path relinking method $\text{PRL}(\tilde{x}, x_{\text{best}}, R_1, R_2)$ (Section 6).

![Figure 1: Outline of the proposed algorithm for SMCP-GUB](image)

8 Computational results

We first prepared eight classes of random instances for SCP \[29\], among which classes G and H were taken from Beasley’s OR Library \[30\] and classes I–N were newly generated in similar manner. Each class has five instances; we denote instances in class G as G.1, …, G.5, and other instances in classes H–N similarly. Another set of benchmark instances is called RAIL arising from a crew pairing problem in an Italian railway company \[5, 7\]. The summary of these instances are given in Table 1, where the density is defined by $\sum_{i \in M} \sum_{j \in N} a_{ij} / mn$.

For each random instance, we generated four types of SMCP-GUB instances with different values of parameters $d_h$ and $|G_h|$ as shown in Table 2 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 multilayer constraints $b_l$ are random integers taken from interval [1, 5].

We compared the proposed algorithm with two recent MIP solvers called CPLEX12.6 and Gurobi5.6.2 and a local search solver called LocalSolver3.1. LocalSolver3.1 is not the latest version, but it performs better than more recent version 4.0 for the benchmark instances. We also compared a 3-flip neighborhood local search algorithm \[8\] for SCP instances. These solvers
Table 1: The benchmark instances for SCP and time limits for the tested algorithms

| instance | #est. | #vars. | density | cost range | MIP solver | heuristics |
|----------|-------|--------|---------|------------|-------------|------------|
| G.1–G.5 | 1000  | 10,000 | 2.0%    | [1,100]    | 3600s      | 600s       |
| H.1–H.5 | 1000  | 10,000 | 5.0%    | [1,100]    | 3600s      | 600s       |
| I.1–I.5 | 1000  | 50,000 | 1.0%    | [1,100]    | 3600s      | 600s       |
| J.1–J.5 | 1000  | 100,000| 1.0%    | [1,100]    | 3600s      | 600s       |
| K.1–K.5 | 2000  | 100,000| 0.5%    | [1,100]    | 7200s      | 1200s      |
| L.1–L.5 | 2000  | 200,000| 0.5%    | [1,100]    | 7200s      | 1200s      |
| M.1–M.5 | 5000  | 1,000,000| 0.25% | [1,100]    | 18,000s    | 3000s      |
| N.1–N.5 | 5000  | 1,000,000| 0.25% | [1,100]    | 18,000s    | 3000s      |
| RAIL507 | 507   | 63,009 | 1.3%    | [1,2]      | 3600s      | 600s       |
| RAIL516 | 516   | 47,311 | 1.3%    | [1,2]      | 3600s      | 600s       |
| RAIL582 | 582   | 55,515 | 1.2%    | [1,2]      | 3600s      | 600s       |
| RAIL2536| 2536  | 1,081,841| 0.4% | [1,2]    | 18,000s    | 3000s      |
| RAIL2586| 2586  | 920,683 | 0.3%    | [1,2]      | 18,000s    | 3000s      |
| RAIL4284| 4284  | 1,092,610| 0.2% | [1,2]    | 18,000s    | 3000s      |
| RAIL4872| 4872  | 968,672 | 0.2%    | [1,2]      | 18,000s    | 3000s      |

Table 2: Four types of benchmark instances for SMCP-GUB \((d_h/|G_h|))

| instance | type1 | type2 | type3 | type4 |
|----------|-------|-------|-------|-------|
| G.1–G.5 | 1/10  | 10/100| 5/100 | 50/100|
| H.1–H.5 | 1/10  | 10/100| 5/50  | 50/100|
| I.1–I.5 | 1/50  | 10/500| 5/50  | 50/500|
| J.1–J.5 | 1/50  | 10/500| 5/50  | 50/500|
| K.1–K.5 | 1/50  | 10/500| 5/50  | 50/500|
| L.1–L.5 | 1/50  | 10/500| 5/50  | 50/500|
| M.1–M.5 | 1/50  | 10/500| 5/50  | 50/500|
| N.1–N.5 | 1/100 | 10/1000| 5/100 | 50/1000|

and the proposed algorithm were tested on a Mac Pro desktop computer with two 2.66 GHz Intel Xeon (six cores) processors and were run on a single thread with time limits shown in Table 1.

We first compared the proposed algorithm with different evaluation schemes of variables: (i) the Lagrangian cost \(\tilde{c}_j(u)\), (ii) the normalized Lagrangian score \(\rho_j\), and (iii) the pseudo-Lagrangian score \(\phi_j\). We also tested the proposed algorithm without the size reduction mechanism.

Table 3 shows the average objective values of the proposed algorithm for each class of SMCP-GUB instances. The third column “\(z_{LP}\)” shows the optimal values of LP relaxation, and the fourth column “w/o” shows the proposed algorithm without the size reduction mechanism. The fifth, sixth, and seventh columns show the results of the proposed algorithm with different evaluation schemes. The best upper bounds among the compared settings are highlighted in bold. We observe that the proposed algorithm with the Lagrangian cost \(\tilde{c}_j(u)\) performs much worse than the algorithm without the size reduction mechanism for types 1 and 2, which indicates that the Lagrangian cost \(\tilde{c}_j(u)\) does not evaluate the promising variables properly for these instances. The proposed algorithm with the normalized Lagrangian score \(\rho_j\) performs much better than the algorithm with the Lagrangian cost \(\tilde{c}_j(u)\) for types 1 and 2, while they show almost the same performance for types 3 and 4. This is because the normalized Lagrangian score \(\rho_j\) takes almost the same value as the Lagrangian cost \(\tilde{c}_j(u)\) for types 3 and 4. We also
Table 3: Computational results of the proposed algorithm with different evaluation schemes of variables for SMCP-GUB

| instance | $z_{LP}$  | w/o | score $\tilde{c}_j(u)$ | score $\rho_j$ | score $\phi_j$ |
|----------|----------|-----|----------------------|----------------|----------------|
| type1    |          |     |                      |                |
| G.1–G.5 | 1715.95  | 2357.4 | 2344.8 | 2356.6 | 2319.4 |
| H.1–H.5 | 395.62   | 603.6 | 602.6 | 601.2 | 604.2 |
| I.1–I.5 | 2845.47  | 3915.6 | 581.0 | 3941.4 | 3808.0 |
| J.1–J.5 | 1466.03  | 2033.0 | 4194.8 | 2020.0 | 1931.6 |
| K.1–K.5 | 5668.91  | 8028.8 | 11873.8 | 8112.8 | 7717.8 |
| L.1–L.5 | 2959.29  | 4278.8 | 8456.8 | 4158.2 | 4002.6 |
| M.1–M.5 | 5476.58  | 8672.0 | 19035.2 | 8369.4 | 7776.2 |
| N.1–N.5 | 4780.91  | 8101.1 | 20662.6 | 7023.4 | 7023.4 |
| type2    |          |     |                      |                |
| G.1–G.5 | 1508.86  | 1905.8 | 1900.2 | 1891.4 | 1902.8 |
| H.1–H.5 | 367.76   | 508.6 | 504.6 | 502.8 | 502.6 |
| I.1–I.5 | 2702.67  | 3549.0 | 5075.8 | 3553.8 | 3446.2 |
| J.1–J.5 | 1393.72  | 1879.4 | 3471.4 | 1835.6 | 1793.8 |
| K.1–K.5 | 5394.32  | 7432.2 | 11530.6 | 7375.6 | 7053.6 |
| L.1–L.5 | 2820.97  | 3949.4 | 7483.4 | 3833.2 | 3664.6 |
| M.1–M.5 | 5244.54  | 8039.6 | 16770.6 | 7664.0 | 7198.0 |
| N.1–N.5 | 4619.25  | 7694.6 | 19565.8 | 7019.2 | 6657.2 |
| type3    |          |     |                      |                |
| G.1–G.5 | 708.03   | 765.0 | 762.2 | 762.2 | 762.2 |
| H.1–H.5 | 190.16   | 212.0 | 211.2 | 211.2 | 211.8 |
| I.1–I.5 | 934.96   | 1125.6 | 1152.0 | 1132.0 | 1105.4 |
| J.1–J.5 | 547.93   | 652.4 | 640.8 | 641.4 | 639.8 |
| K.1–K.5 | 1889.22  | 2313.4 | 2274.2 | 2263.2 | 2252.2 |
| L.1–L.5 | 1104.29  | 1339.2 | 1259.2 | 1257.8 | 1254.8 |
| M.1–M.5 | 2038.85  | 2708.2 | 2597.4 | 2571.2 | 2544.8 |
| N.1–N.5 | 1747.88  | 2495.4 | 2347.8 | 2310.2 | 2228.8 |
| type4    |          |     |                      |                |
| G.1–G.5 | 691.14   | 732.8 | 730.2 | 730.2 | 730.8 |
| H.1–H.5 | 187.71   | 204.4 | 203.6 | 203.6 | 203.2 |
| I.1–I.5 | 917.48   | 1075.6 | 1060.8 | 1064.2 | 1059.4 |
| J.1–J.5 | 539.06   | 625.6 | 615.4 | 615.4 | 611.2 |
| K.1–K.5 | 1853.17  | 2217.0 | 2172.6 | 2175.8 | 2161.8 |
| L.1–L.5 | 1088.47  | 1291.2 | 1259.2 | 1257.8 | 1254.4 |
| M.1–M.5 | 2051.53  | 2592.2 | 2487.0 | 2495.2 | 2459.0 |
| N.1–N.5 | 1724.53  | 2337.6 | 2244.2 | 2235.2 | 2170.2 |

observed that the proposed algorithm with the pseudo-Lagrangian score $\phi_j$ performs better than the algorithm with the normalized Lagrangian score $\rho_j$ for most of the tested instances.

Table 4 shows the average objective values of the proposed algorithm for each class of SCP instances, in which we omit the results for the normalized Lagrangian score $\rho_j$ because it takes exactly the same value as the Lagrangian cost $\tilde{c}_j(u)$ for SCP instances. We observe that the proposed algorithm with the Lagrangian cost $\tilde{c}_j(u)$ and the pseudo-Lagrangian score $\phi_j$ performs better than the algorithm without the size reduction mechanism, and the algorithm with the Lagrangian cost $\tilde{c}_j(u)$ performs best for the RAIL instances.

We next compared the proposed algorithm with the above-mentioned recent solvers, where we tested the proposed algorithm with the pseudo-Lagrangian score $\phi_j$. Tables 5 and 6 show the average objective values of the compared algorithms for each class of SMCP-GUB and SCP instances, respectively, where the results with asterisks “∗” indicate that the obtained feasible solutions were proven to be optimal. We observe that the proposed algorithm performs better than CPLEX12.6, Gurobi5.6.2 and LocalSolver3.1 for all types of SMCP-GUB instances and classes G–N of SCP instances while it performs worse than CPLEX12.6 and Gurobi5.6.2 for class RAIL of SCP instances. We also observe that the 3-flip neighborhood local search algorithm [S]...
Table 4: Computational results of the proposed algorithm with different evaluation schemes for SCP

| instance | $z_{w,LP}$ | w/o score $\tilde{c}_j(u)$ | score $\tilde{c}_j(u)$ | score $\phi_j$ |
|----------|-----------|-----------------|-----------------|--------------|
| G.1–G.5 | 149.48    | 166.4           | 166.4           | 166.4        |
| H.1–H.5 | 45.67     | 59.6            | 59.6            | 59.6         |
| I.1–I.5 | 138.96    | 160.2           | 159.0           | 158.8        |
| J.1–J.5 | 104.78    | 132.4           | 130.8           | 131.0        |
| K.1–K.5 | 276.66    | 322.0           | 319.2           | 318.4        |
| L.1–L.5 | 209.33    | 267.8           | 263.4           | 264.4        |
| M.1–M.5 | 415.77    | 575.8           | 565.4           | 566.2        |
| N.1–N.5 | 348.92    | 535.4           | 516.8           | 518.8        |
| RAIL507 | 172.14    | 179             | 176             | 178          |
| RAIL516 | 182.00    | 183             | 182             | 183          |
| RAIL582 | 209.71    | 217             | 212             | 216          |
| RAIL2536| 688.39    | 718             | 693             | 716          |
| RAIL2586| 935.92    | 1010            | 965             | 989          |
| RAIL4284| 1054.05   | 1125            | 1080            | 1130         |
| RAIL4872| 1509.63   | 1618            | 1565            | 1591         |

achieves best upper bounds for almost all classes of SCP instances.

These observations indicate that the variable fixing and pricing techniques based on the LP and/or Lagrangian relaxation are greatly affected by the gaps between lower and upper bounds, and they may not work effectively for the instances having large gaps. For such instances, the proposed algorithm with the pseudo-Lagrangian score $\phi_j$ succeeds in evaluating the promising variables properly.

9 Conclusion

In this paper, we considered an extension of the set covering problem (SCP) called the set multicover problem with the generalized upper bound constraints (SMCP-GUB). We developed a 2-flip local search algorithm incorporated with heuristic algorithms to reduce the size of instances, evaluating promising variables by taking account of GUB constraints, and with an adaptive control mechanism of penalty weights that is used to guide the search to visit feasible and infeasible regions alternately. We also developed an efficient implementation of a 2-flip neighborhood search that reduces the number of candidates in the neighborhood without sacrificing the solution quality. To guide the search to visit a wide variety of good solutions, we also introduced an evolutionary approach called the path relinking method that generates new solutions by combining two or more solutions obtained by then. According to computational comparison on benchmark instances with three recent solvers, we can conclude that the proposed algorithm performs quite effectively for instances having large gaps between lower and upper bounds.

We expect that the evaluation scheme of promising variables is also applicable to other combinatorial optimization problems, because the pseudo-Lagrangian score $\phi_j = \tilde{c}_j(w)$ ($j \in N$) can be defined when an adaptive control mechanism of penalty weights $w$ is incorporated in local search, and such an approach has been observed to be effective for many problems.
Table 5: Computational results of the tested solvers and the proposed algorithm for SMCP-GUB

| Instance | \( z \) | CPLEX | Gurobi | LocalSolver | proposed |
|----------|-------|--------|--------|-------------|---------|
| type1    |       |        |        |             |         |
| G.1–G.5 | 1715.95 | 2590.8 | 2629.6 | 4096.2 | 2319.4 |
| H.1–H.5 | 395.62  | 680.8  | 705.4  | 1217.2 | 604.2  |
| I.1–I.5 | 2845.47 | 4757.2 | 4544.6 | 7830.8 | 3808.0 |
| J.1–J.5 | 1466.03 | 2665.4 | 2392.6 | 4744.6 | 1936.1 |
| K.1–K.5 | 5668.91 | 12363.6| 9284.4 | 16235.0| 7717.8 |
| L.1–L.5 | 2595.29 | 7607.4 | 4749.2 | 9904.0 | 4002.6 |
| M.1–M.5 | 5476.58 | 16211.0| 10152.2| 21564.8| 7776.2 |
| N.1–N.5 | 4780.91 | 16970.8| 8966.2 | 21618.8| 7023.4 |
| type2    |       |        |        |             |         |
| G.1–G.5 | 1508.86 | 2053.8 | 2023.8 | 3227.0 | 1902.8 |
| H.1–H.5 | 367.76  | 555.2  | 560.0  | 915.4 | 502.6  |
| I.1–I.5 | 2702.67 | 4382.0 | 3959.2 | 6514.4 | 3446.2 |
| J.1–J.5 | 1393.72 | 3156.2 | 2132.6 | 3837.6 | 1793.8 |
| K.1–K.5 | 5394.32 | 15763.6| 7952.2 | 13647.0| 7053.6 |
| L.1–L.5 | 2820.97 | 15683.0| 4338.4 | 8117.4 | 3664.6 |
| M.1–M.5 | 5244.54 | 36794.8| 8600.8 | 17614.4| 7198.0 |
| N.1–N.5 | 4619.25 | 36970.0| 8154.4 | 17732.6| 6657.2 |
| type3    |       |        |        |             |         |
| G.1–G.5 | 708.03  | 766.6  | 769.8  | 1585.4 | 702.2  |
| H.1–H.5 | 190.16  | 216.4  | 216.4  | 554.8 | 211.8  |
| I.1–I.5 | 934.96  | 1223.6 | 1176.2 | 2326.4 | 1105.4 |
| J.1–J.5 | 547.93  | 707.4  | 669.2  | 1802.8 | 639.8  |
| K.1–K.5 | 1889.22 | 3802.8 | 2530.6 | 4788.8 | 2252.2 |
| L.1–L.5 | 1104.29 | 3318.2 | 1384.6 | 4294.4 | 1302.4 |
| M.1–M.5 | 2083.85 | 8404.4 | 2691.6 | 9575.8 | 2544.8 |
| N.1–N.5 | 1747.88 | 8394.0 | 2597.6 | 10005.4| 2228.8 |
| type4    |       |        |        |             |         |
| G.1–G.5 | 691.14  | 735.2  | 731.8  | 1495.0 | 730.8  |
| H.1–H.5 | 187.71  | 206.8  | 207.2  | 563.4 | 203.2  |
| I.1–I.5 | 917.48  | 1150.0 | 1089.6 | 2109.0 | 1059.4 |
| J.1–J.5 | 539.06  | 644.4  | 629.8  | 1698.4 | 611.2  |
| K.1–K.5 | 1853.17 | 2734.2 | 2231.6 | 4654.4 | 2161.8 |
| L.1–L.5 | 1088.47 | 1566.6 | 1299.6 | 3809.6 | 1254.4 |
| M.1–M.5 | 2051.53 | 7572.6 | 2547.2 | 9909.4 | 2459.0 |
| N.1–N.5 | 1724.53 | 8371.8 | 2339.8 | 9906.8 | 2170.2 |

References

[1] Umetani S, Arakawa M, Yagiura M. A heuristic algorithm for the set multicover problem with generalized upper bound constraints. In: Proceedings of Learning and Intelligent Optimization Conference (LION). Berlin: Springer; 2013. p. 75–80.

[2] Caprara A, Toth P, Fischetti M. Algorithms for the set covering problem. Ann Oper Res 2000; 98: 353–71.

[3] Umetani S., Yagiura M. Relaxation heuristics for the set covering problem. J Oper Res Soc Jpn 2007; 50: 350–75.

[4] Beasley JE. A Lagrangian heuristic for set-covering problems. Nav Res Logist 1990; 37: 151–64.

[5] Caprara A, Fischetti M, Toth P. A heuristic method for the set covering problem. Oper Res 1999; 47: 730–43.
Table 6: Computational results of the tested solvers and the proposed algorithm for SCP

| instance | $z_{LP}$ | CPLEX | Gurobi | LocalSolver | Yagiura et al. | proposed |
|----------|----------|--------|--------|-------------|----------------|----------|
| G.1–G.5 | 149.48   | 166.6  | 166.6  | 316.6       | 166.4          | 166.4    |
| H.1–H.5 | 45.67    | 60.2   | 60.2   | 169.2       | 59.6           | 59.6     |
| L.1–L.5 | 138.96   | 162.4  | 162.4  | 403.4       | 158.0          | 158.8    |
| K.1–K.5 | 276.66   | 325.8  | 325.2  | 828.2       | 313.2          | 318.4    |
| L.1–L.5 | 209.33   | 285.4  | 273.6  | 918.6       | 258.6          | 264.4    |
| M.1–M.5 | 415.77   | 637.2  | 609.8  | 2163.8      | 550.2          | 566.2    |
| N.1–N.5 | 348.92   | 753.0  | 577.4  | 2382.2      | 503.8          | 518.8    |

RAIL507  172.14  *174  *174  187  174  178
RAIL516  182.00  *182  *182  189  182  183
RAIL582  209.71  *211  *211  227  21  216
RAIL2536 688.39  *689  *689  729  691  716
RAIL2586 935.92  957  972  1012 947  989
RAIL4284 1054.05 1077  1085  1148 1064 1130
RAIL4872 1509.63 1549  1559  1651 1531 1591

[6] Caserta M. Tabu search-based metaheuristic algorithm for large-scale set covering problems. In: Gutjahr WJ, Hartl RF, Reimann M., editors. Metaheuristics: Progress in Complex Systems Optimization, Berlin: Springer; 2007, p. 43–63.

[7] Ceria S, Nobili P, Sassano A. A Lagrangian-based heuristic for large-scale set covering problems. Math Program 1998; 81: 215–88.

[8] Yagiura M, Kishida M, Ibaraki T. A 3-flip neighborhood local search for the set covering problem. Eur J Oper Res 2006; 172: 472–99.

[9] 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 2009; 5: 185–202.

[10] Boros E, Ibaraki T, Ichikawa H, Nonobe K, Uno T, Yagiura M. Heuristic approaches to the capacitated square covering problem. Pac J Optim 2005; 1: 465–90.

[11] Farahani RZ, Asgari N, Heidari N, Hosseininia M, Goh M. Covering problems in facility location: A review. Comput Ind Eng 2012; 62: 368–407.

[12] Boros E, Hammer PL, Ibaraki T, Kogan A, Mayorz E, Muchnik I. An implementation of logical analysis of data. IEEE Trans Knowl Data Eng 2000; 12: 292–306.

[13] Bettinelli A, Ceselli A, Righini G. A branch-and-price algorithm for the multi-depot heterogeneous-fleet pickup and delivery problem with soft time windows. Math Program 2014; 6: 171–97.

[14] Choi E, Tcha DW. A column generation approach to the heterogeneous fleet vehicle routing problem. Comput Oper Res 2007; 34: 2080–95.

[15] Kohl N, Karisch S. Airline crew rostering: Problem types, modeling, and optimization. Ann Oper Res 2004; 127: 223–57.

[16] Caprara A, Monaci M, Toth P. Models and algorithms for a staff scheduling problem. Math Program 2003; B98: 445–76.
A Proof of Lemma 1

We show that \( \Delta z_{j_1,j_2}(x, w) \geq 0 \) holds if \( x_{j_1} = x_{j_2} = 1 \). First, we consider the case with \( x_{j_1} = x_{j_2} = 1 \).

By the assumption of the lemma,

\[
\Delta z_{j}^{\downarrow}(x, w) = -c_j + \sum_{i \in (M_L(x) \cup M_E(x)) \cap S_j} w_i \geq 0
\]  

holds for both \( j = j_1 \) and \( j_2 \). Then we have

\[
\Delta z_{j_1,j_2}(x, w) = \Delta z_{j_1}^{\downarrow}(x, w) + \Delta z_{j_2}^{\uparrow}(x, w) + \sum_{i \in M_+(x) \cap S_{j_1} \cap S_{j_2}} w_i \geq 0,
\]  

holds.
where \( M_+(x) = \{ i \in M \mid \sum_{j \in N} a_{ij} x_j = b_i + 1 \} \). Next, we consider the case with \( x_{j1} = x_{j2} = 0 \). By the assumption of the lemma, for both \( j = j_1 \) and \( j_2 \),

\[
\Delta \hat{z}^j_{j_1}(x, w) = c_j - \sum_{i \in M_L(x) \cap S_j} w_i \geq 0
\]  

(16)

holds unless \( j \) belongs a block \( G_h \) satisfying \( \sum_{l \in G_h} x_l = d_h \). Then we have

\[
\Delta \hat{z}_{j_1,j_2}(x, w) = \Delta \hat{z}^j_{j_1}(x, w) + \Delta \hat{z}^j_{j_2}(x, w) + \sum_{i \in M_-(x) \cap S_{j_1} \cap S_{j_2}} w_i \geq 0,
\]  

(17)

where \( M_-(x) = \{ i \in M \mid \sum_{j \in N} a_{ij} x_j = b_i - 1 \} \).

B Proof of Lemma 2

We assume that neither the conditions (i) nor (ii) is satisfied and \( \Delta \hat{z}_{j_1,j_2}(x, w) < 0 \) holds. Then, for (ii) is assumed to be unsatisfied, \( M_E(x) \cap S_{j_2} \cap S_{j_2} = \emptyset \) holds, and hence we have \( \Delta \hat{z}_{j_1,j_2}(x, w) = \Delta \hat{z}^j_{j_1}(x, w) + \Delta \hat{z}^j_{j_2}(x, w) < 0 \), which implies \( \Delta \hat{z}^j_{j_1}(x, w) < 0 \) or \( \Delta \hat{z}^j_{j_2}(x, w) < 0 \). If \( \Delta \hat{z}^j_{j_1}(x, w) < 0 \) holds, then the algorithm obtains an improved solution by flipping \( x_{j_1} = 1 \rightarrow 0 \). If \( \Delta \hat{z}^j_{j_2}(x, w) < 0 \) holds, the algorithm obtains an improved solution by flipping \( x_{j_2} = 0 \rightarrow 1 \), because \( j_1 \) and \( j_2 \) belong to the same block satisfying \( \sum_{j \in E} x_j < d_h \) or to different blocks, and in the latter case, \( j_2 \) belongs a block \( G_h \) satisfying \( \sum_{j \in G_h} x_j < d_h \) by the assumption that we only consider solutions that satisfy the GUB constraints, including the solution obtained after flipping \( j_1 \) and \( j_2 \). Both cases contradict the assumption that \( x \) is locally optimal with respect to \( NB_1(x) \).

C Proof of Lemma 3

We have \( \Delta \hat{z}_{j_1,j_2}(x, w) = \Delta \hat{z}^j_{j_1}(x, w) + \Delta \hat{z}^j_{j_2}(x, w) \), because \( M_E(x) \cap S_{j_2} \cap S_{j_2} = \emptyset \) holds. Then we have \( \min_{j \in G_h} \Delta \hat{z}^j_{j_1}(x, w) + \min_{j \in G_h} \Delta \hat{z}^j_{j_2}(x, w) \leq \Delta \hat{z}^j_{j_1}(x, w) + \Delta \hat{z}^j_{j_2}(x, w) = \Delta \hat{z}_{j_1,j_2}(x, w) \), because \( \min_{j \in G_h} \Delta \hat{z}^j_{j_1}(x, w) \leq \Delta \hat{z}^j_{j_1}(x, w) \) and \( \min_{j \in G_h} \Delta \hat{z}^j_{j_2}(x, w) \leq \Delta \hat{z}^j_{j_2}(x, w) \) hold. Let \( j^*_1 = \arg \min_{j \in G_h} \Delta \hat{z}^j_{j_1}(x, w) \) and \( j^*_2 = \arg \min_{j \in G_h} \Delta \hat{z}^j_{j_2}(x, w) \). Then we have \( \Delta \hat{z}_{j^*_1,j^*_2}(x, w) = \Delta \hat{z}^j_{j^*_1}(x, w) + \Delta \hat{z}^j_{j^*_2}(x, w) - \sum_{i \in M_E(x) \cap S_{j^*_1} \cap S_{j^*_2}} w_i < \Delta \hat{z}^j_{j_1}(x, w) + \Delta \hat{z}^j_{j_2}(x, w) < 0 \).

D Efficient incremental evaluation of solutions in 2-FNLS

We first consider the case that the current solution \( x \) moves to \( x' \) by flipping \( x_j = 0 \rightarrow 1 \). Then, the algorithm first updates \( s_i(x) \) \((i \in S_j)\) in \( O(|S_j|) \) time by \( s_i(x') \leftarrow s_i(x) + 1 \), and then updates \( \Delta p_i(x, w) \) and \( \Delta p_i(x, w) \) \((i \in N_i, i \in S_j)\) in \( O(\sum_{i \in S_j} |N_i|) \) time by \( \Delta p_i(x', w) \leftarrow \Delta p_i(x, w) - w_i \) if \( s_i(x') = b_i + 1 \) holds and \( \Delta p_i(x', w) \leftarrow \Delta p_i(x, w) - w_i \) if \( s_i(x') = b_i \) holds. Similarly, we consider the other case that the current solution \( x \) moves to \( x' \) by flipping \( x_j = 1 \rightarrow 0 \). Then, the algorithm first updates \( s_i(x) \) \((i \in S_j)\) in \( O(|S_j|) \) time by \( s_i(x') \leftarrow s_i(x) - 1 \), and then updates \( \Delta p_i(x, w) \) and \( \Delta p_i(x, w) \) \((i \in N_i, i \in S_j)\) in \( O(\sum_{i \in S_j} |N_i|) \) time by \( \Delta p_i(x', w) \leftarrow \Delta p_i(x, w) + w_i \) if \( s_i(x') = b_i \) holds and \( \Delta p_i(x', w) \leftarrow \Delta p_i(x, w) + w_i \) if \( s_i(x') = b_i + 1 \) holds.