Exploiting variable associations to configure efficient local search algorithms in large-scale binary integer programs*

Shunji Umetani†

April 29, 2016

Abstract

We present a data mining approach for reducing the search space of local search algorithms in a class of binary integer programs including the set covering and partitioning problems. We construct a $k$-nearest neighbor graph by extracting variable associations from the instance to be solved, in order to identify promising pairs of flipping variables in the neighborhood search. We also develop a 4-flip neighborhood local search algorithm that flips four variables alternately along 4-paths or 4-cycles in the $k$-nearest neighbor graph. We incorporate an efficient incremental evaluation of solutions and an adaptive control of penalty weights into the 4-flip neighborhood local search algorithm. Computational comparison with the latest solvers shows that our algorithm performs effectively for large-scale set covering and partitioning problems.

keyword: combinatorial optimization, heuristics, set covering problem, set partitioning problem, local search

1 Introduction

The set covering problem (SCP) and set partitioning problem (SPP) are representative combinatorial optimization problems that have many real-world applications, such as crew scheduling [6, 24, 30], vehicle routing [2, 14, 23], facility location [12, 21] and logical analysis of data [11, 22]. Real-world applications of SCP and SPP are comprehensively reviewed in [4, 18].

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 $c_j \in \mathbb{R}$ 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. We say that $X \subseteq N$ is a partition of $M$ if $\bigcup_{j \in X} S_j = M$ and $S_{j_1} \cap S_{j_2} = \emptyset$ hold for all $j_1, j_2 \in X$. The goals of SCP and SPP are to find a minimum cost cover and a partition $X$ of $M$, respectively. In this paper, we consider the following class of binary integer programs (BIPs) including SCP and SPP:

$$\begin{align*}
\text{minimize} \quad & \sum_{j \in N} c_j x_j \\
\text{subject to} \quad & \sum_{j \in N} a_{ij} x_j \geq b_i, \quad i \in M_G, \\
& \sum_{j \in N} a_{ij} x_j \leq b_i, \quad i \in M_L, \\
& \sum_{j \in N} a_{ij} x_j = b_i, \quad i \in M_E, \\
& x_j \in \{0, 1\}, \quad j \in N,
\end{align*}$$

(1)

* A preliminary version of this paper was presented in [39].
† Graduate School of Information Science and Technology, Osaka University, 1-5 Yamadaoka, Suita, Osaka, 565-0871, Japan. umetani@ist.osaka-u.ac.jp
where $a_{ij} \in \{0,1\}$ and $b_i \in \mathbb{Z}_+$ ($\mathbb{Z}_+$ is the set of non-negative integer values). We note that $a_{ij} = 1$ 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 vector $a_j = (a_{ij_1}, \ldots, a_{iM})^T$ of the matrix $(a_{ij})$ represents the corresponding subset $S_j$ by $S_j = \{ i \in M_G \cup M_L \cup M_E \mid a_{ij} = 1 \}$, and the vector $x$ also represents the corresponding cover (or partition) $X$ by $X = \{ j \in N \mid x_j = 1 \}$. For notational convenience, for each $i \in M_G \cup M_L \cup M_E$, let $N_i = \{ j \in N \mid a_{ij} = 1 \}$ be the index set of subsets $S_j$ that contain the elements $i$ and let $s_i(x) = \sum_{j \in N} a_{ij}x_j$ be the left-hand side of the $i$th constraint.

The SCP and SPP are known to be NP-hard in the strong sense, and no polynomial time approximation scheme (PTAS) exists unless $P = NP$. However, worst case performance analysis does not necessarily represent the experimental performance in practice. Continuous development of mathematical programming has much improved the performance of heuristic algorithms and this has been accompanied by advances in computing machinery. Many efficient exact and heuristic algorithms for large-scale SCP and SPP instances have been developed [3, 5, 7, 10, 13, 15, 16, 17, 19, 28, 37, 42, 44], many of which are based on variable fixing techniques that reduce the search space to be explored by using the optimal values obtained by linear programming (LP) and/or Lagrangian relaxation as lower bounds. However, many large-scale SCP and SPP instances still remain unsolved because there is little hope of closing the large gap between the lower and upper bounds of the optimal values. In particular, the equality constraints of SPP often make variable fixing techniques less effective because they often prevent solutions from containing highly evaluated variables together. In this paper, we consider an alternative approach for extracting useful features from the instance to be solved with the aim of reducing the search space of local search algorithms for large-scale SCP and SPP instances.

In the design of local search algorithms for large-scale combinatorial problems, as the instance size increases, improving the computational efficiency becomes more effective than using sophisticated search strategies. The quality of locally optimal solutions typically improves if a larger neighborhood is used. However, the computation time of searching the neighborhood also increases exponentially. To overcome this, extensive research has investigated ways to efficiently implement neighborhood search, which can be broadly classified into three types: (i) reducing the number of candidates in the neighborhood [33, 35, 43, 44], (ii) evaluating solutions by incremental computation [29, 43, 40, 41], and (iii) reducing the number of variables to be considered by using linear programming and/or Lagrangian relaxation [15, 19, 38, 44].

To suggest an alternative, we develop a data mining approach for reducing the search space of local search algorithms. That is, we construct a $k$-nearest neighbor graph by extracting variable associations from the instance to be solved in order to identify promising pairs of swapping variables in the large neighborhood search. We also develop a 4-flip neighborhood local search algorithm that flips four variables alternately along 4-paths or 4-cycles in the $k$-nearest neighbor graph (Section 3). We incorporate an efficient incremental evaluation of solutions (Section 2) and an adaptive control of penalty weights (Section 4) into the 4-flip neighborhood local search algorithm.

## 2 2-flip neighborhood local search

Local search (LS) starts from an initial solution $x$ and then iteratively replaces $x$ with a better solution $x'$ in the neighborhood $NB(x)$ until no better solution is found in $NB(x)$. For some positive integer $r$, let the $r$-flip neighborhood $NB_r(x)$ be the set of solutions obtainable by flipping at most $r$ variables in $x$. We first develop a 2-flip neighborhood local search (2-FNLS) algorithm as a basic component of our algorithm. In order to improve efficiency, the 2-FNLS first searches $NB_1(x)$, and then searches $NB_2(x) \setminus NB_1(x)$ only if $x$ is locally optimal with respect to $NB_1(x)$.
The BIP 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. We accordingly consider the following formulation of a BIP that allows violations of the constraints and introduce over and under penalty functions with penalty weight vectors $w^+, w^- \in \mathbb{R}_+^n$:

$$
\begin{align*}
\text{minimize} & \quad z(x) = \sum_{j \in N} c_j x_j + \sum_{i \in M_L \cup M_E} w^+_i y^+_i + \sum_{i \in M_G \cup M_E} w^-_i y^-_i \\
\text{subject to} & \quad \sum_{j \in N} a_{ij} x_j + y^-_i \geq b_i, \quad i \in M_G, \\
& \quad \sum_{j \in N} a_{ij} x_j - y^+_i \leq b_i, \quad i \in M_L, \\
& \quad \sum_{j \in N} a_{ij} x_j + y^-_i - y^+_i = b_i, \quad i \in M_E, \\
& \quad x_j \in \{0, 1\}, \quad j \in N, \\
& \quad y^+_i \geq 0, \quad i \in M_L \cup M_E, \\
& \quad y^-_i \geq 0, \quad i \in M_G \cup M_E.
\end{align*}
$$

For a given $x \in \{0, 1\}^n$, we can easily compute optimal $y^+_i = |s_i(x) - b_i|_+$ and $y^-_i = |b_i - s_i(x)|_+$, where we denote $|x|_+ = \max\{x, 0\}$.

Because the region searched by a single application of LS is limited, LS is usually applied many times. When a locally optimal solution is found, a standard strategy is to update the penalty weights and to resume LS from the obtained locally optimal solution. We accordingly evaluate solutions by using an alternative function $\tilde{z}(x)$, where the original penalty weight vectors $w^+, w^-$ are replaced by $\tilde{w}^+, \tilde{w}^-$, and these are adaptively controlled during the search (see Section 3 for details).

We first describe how 2-FNLS is used to search $NB_1(x)$, which is called the 1-flip neighborhood. Let $\Delta \tilde{z}_j^\top(x)$ and $\Delta \tilde{z}_j^\downarrow(x)$ denote the increases in $\tilde{z}(x)$ due to flipping $x_j = 0 \rightarrow 1$ and $x_j = 1 \rightarrow 0$, respectively. 2-FNLS first searches for an improved solution by flipping $x_j = 0 \rightarrow 1$ for $j \in N \setminus X$. If an improved solution is found, it chooses $j$ that has the minimum value of $\Delta \tilde{z}^\top(x)$, otherwise it searches for an improved solution by flipping $x_j = 1 \rightarrow 0$ for $j \in X$.

We next describe how 2-FNLS is used to search $NB_2(x) \setminus NB_1(x)$, which is called the 2-flip neighborhood. We derive conditions that reduce the number of candidates in $NB_2(x) \setminus NB_1(x)$ without sacrificing the solution quality by expanding the results as shown in [14]. Let $\Delta \tilde{z}_{j_1,j_2}(x)$ denote the increase in $\tilde{z}(x)$ due to simultaneously flipping the values of $x_{j_1}$ and $x_{j_2}$.

**Lemma 1.** If a solution $x$ is locally optimal with respect to $NB_1(x)$, then $\Delta \tilde{z}_{j_1,j_2}(x) < 0$ holds only if $x_{j_1} \neq x_{j_2}$.

**Proof.** See Appendix A \(\square\)

Based on this lemma, we consider only the set of solutions obtainable by simultaneously flipping $x_{j_1} = 1 \rightarrow 0$ and $x_{j_2} = 0 \rightarrow 1$. We now define

$$
\Delta \tilde{z}_{j_1,j_2}(x) = \Delta \tilde{z}_j^\top(x) + \Delta \tilde{z}_j^\downarrow(x) - \sum_{i \in S(x) \cap (M_L \cup M_E)} \tilde{w}_i^+ - \sum_{i \in S(x) \cap (M_G \cup M_E)} \tilde{w}_i^-,
$$

where $S(x) = \{i \in S_{j_1} \cap S_{j_2} \mid s_i(x) = b_i\}$. By Lemma 1, the 2-flip neighborhood can be restricted to the set of solutions satisfying $x_{j_1} \neq x_{j_2}$ and $S(x) \neq \emptyset$. However, it might not be possible to search this set efficiently without first extracting it. We thus construct a neighbor list that stores promising pairs of variables $x_{j_1}$ and $x_{j_2}$ for efficiency (see Section 3 for details).
To increase the efficiency of 2-FNLS, we decompose the neighborhood \( \text{NB}_2(\mathbf{x}) \) into a number of sub-neighborhoods. Let \( \text{NB}^{(j_1)}_2(\mathbf{x}) \) denote the subset of \( \text{NB}_2(\mathbf{x}) \) obtainable by flipping \( x_{j_1} = 1 \to 0 \). 2-FNLS searches \( \text{NB}^{(j_1)}_2(\mathbf{x}) \) for each \( j_1 \in X \) in ascending order of \( \Delta \tilde{z}^{j_1}(\mathbf{x}) \). If an improved solution is found, the pair \( j_1 \) and \( j_2 \) that has the minimum value of \( \Delta \tilde{z}_{j_1,j_2}(\mathbf{x}) \) among \( \text{NB}^{(j_1)}_2(\mathbf{x}) \) is selected. 2-FNLS is formally described as follows.

**Algorithm 2-FNLS**(\( \mathbf{x}, \mathbf{\tilde{w}}^+, \mathbf{\tilde{w}}^- \))

**Input:** A solution \( \mathbf{x} \) and penalty weight vectors \( \mathbf{\tilde{w}}^+ \) and \( \mathbf{\tilde{w}}^- \).

**Output:** A solution \( \mathbf{x} \).

**Step 1:** If \( I_1^r(\mathbf{x}) = \{ j \in N \setminus X \mid \Delta \tilde{z}_j^r(\mathbf{x}) < 0 \} \neq \emptyset \) holds, choose \( j \in I_1^r(\mathbf{x}) \) that has the minimum value of \( \Delta \tilde{z}_j^r(\mathbf{x}) \), set \( x_j \leftarrow 1 \), and return to Step 1.

**Step 2:** If \( I_1^r(\mathbf{x}) = \{ j \in X \mid \Delta \tilde{z}_j^r(\mathbf{x}) < 0 \} \neq \emptyset \) holds, choose \( j \in I_1^r(\mathbf{x}) \) that has the minimum value of \( \Delta \tilde{z}_j^r(\mathbf{x}) \), set \( x_j \leftarrow 0 \), and return to Step 2.

**Step 3:** For each \( j_1 \in X \) in ascending order of \( \Delta \tilde{z}^{j_1}(\mathbf{x}) \), if \( I_2^{(j_1)}(\mathbf{x}) = \{ j_2 \in N \setminus X \mid \Delta \tilde{z}_{j_1,j_2}(\mathbf{x}) < 0 \} \neq \emptyset \) holds, choose \( j_2 \in I_2^{(j_1)}(\mathbf{x}) \) that has the minimum value of \( \Delta \tilde{z}_{j_1,j_2}(\mathbf{x}) \) and set \( x_{j_1} \leftarrow 0 \) and \( x_{j_2} \leftarrow 1 \). If the current solution \( \mathbf{x} \) has been updated at least once in Step 3, return to Step 1, otherwise output \( \mathbf{x} \) and exit.

If implemented naively, 2-FNLS requires \( O(\sigma) \) time to compute the value of the evaluation function \( \tilde{z}(\mathbf{x}) \) for the current solution \( \mathbf{x} \), where \( \sigma = \sum_{i \in M} \sum_{j \in N} a_{ij} \) denote the number of non-zero elements in the constraint matrix \( (a_{ij}) \). This computation is quite expensive if we evaluate the neighbor solutions of the current solution \( \mathbf{x} \) independently. To overcome this, we first develop a standard incremental evaluation of \( \Delta \tilde{z}_j^r(\mathbf{x}) \) and \( \Delta \tilde{z}^{j_1}(\mathbf{x}) \) in \( O(|S_j|) \) time by keeping the values of the left-hand side of constraints \( s_i(\mathbf{x}) \) \( (i \in M_G \cup M_L \cup M_E) \) in memory. We further develop an improved incremental evaluation of \( \Delta \tilde{z}_j^r(\mathbf{x}) \) and \( \Delta \tilde{z}^{j_1}(\mathbf{x}) \) in \( O(1) \) time by keeping additional auxiliary data in memory (see Appendix B for details). By using this, 2-FNLS is also able to evaluate \( \Delta \tilde{z}_{j_1,j_2}(\mathbf{x}) \) in \( O(|S_j|) \) time by using (3).

### 3 Exploiting variable associations

It is known that the quality of locally optimal solutions improves if a larger neighborhood is used. However, the computation time to search the neighborhood \( \text{NB}_r(\mathbf{x}) \) also increases exponentially with \( r \), since \( |\text{NB}_r(\mathbf{x})| = O(n^r) \) generally holds. A large amount of computation time is thus needed in practice in order to scan all candidates in \( \text{NB}_2(\mathbf{x}) \) for large-scale instances with millions of variables. To overcome this, we develop a data mining approach that identifies promising pairs of flipping variables in \( \text{NB}_2(\mathbf{x}) \) by extracting variable associations from the instance to be solved using only a small amount of computation time.

Based on Lemma 1, the 2-flip neighborhood can be restricted to the set of solutions satisfying \( x_{j_1} \neq x_{j_2} \) and \( S(\mathbf{x}) \neq \emptyset \). We further observe from (3) that it is favorable to select pairs of flipping variables \( x_{j_1} \) and \( x_{j_2} \) which gives a larger size \( |S_{j_1} \cap S_{j_2}| \) in order to obtain \( \Delta \tilde{z}_{j_1,j_2}(\mathbf{x}) < 0 \). Based on this observation, we keep a limited set of pairs of variables \( x_{j_1} \) and \( x_{j_2} \) for which \( |S_{j_1} \cap S_{j_2}| \) is large in memory, and we call this the neighbor list (Figure 1). We note that \( |S_{j_1} \cap S_{j_2}| \) represents a certain kind of similarity between the subsets \( S_{j_1} \) and \( S_{j_2} \) (or column vectors \( a_{j_1} \) and \( a_{j_2} \) of the constraint matrix \( (a_{ij}) \)) and we keep the \( k \)-nearest neighbors for each subset \( S_j \) \( (j \in N) \) in the neighbor list.
For each variable $x_{j_1}$ ($j_1 \in N$), we first enumerate $x_{j_2}$ ($j_2 \in N$) satisfying $j_2 \neq j_1$ and $S_{j_1} \cap S_{j_2} \neq \emptyset$ to generate the set $L[j_1]$, and store the first $\min\{|N^{(j_1)}|, \alpha|M_G \cup M_L \cup M_L|\}$ variables $x_{j_2}$ ($j_2 \in L[j_1]$) in descending order of $|S_{j_1} \cap S_{j_2}|$ in the $j_1$th row of the neighbor list, where $N^{(j_1)} = \{j_2 \in N \mid j_2 \neq j_1, S_{j_1} \cap S_{j_2} \neq \emptyset\}$ and $\alpha$ is a program parameter that we set to five. Let $L'[j_1]$ be the set of variables $x_{j_2}$ stored in the $j_1$th row of the neighbor list. We then reduce the number of candidates in NB2($x$) by restricting the pairs of flipping variables $x_{j_1}$ and $x_{j_2}$ to pairs in the neighbor list $j_1 \in X$ and $j_2 \in (N \setminus X) \cap L'[j_1]$.

We note that it is still expensive to construct the whole neighbor list for large-scale instances with millions of variables. To overcome this, we develop a lazy construction algorithm for the neighbor list. That is, 2-FNLS starts from an empty neighbor list and generates the $j_1$th row of the neighbor list $L'[j_1]$ only when 2-FNLS searches NB2($x$) for the first time.

Figure 1: Example of a neighbor list

Figure 2: Example of the $k$-nearest neighbor graph

We can treat the neighbor list as an adjacency-list representation of a directed graph, and represent associations between variables by a corresponding directed graph called the $k$-nearest neighbor graph (Figure 2). Using the $k$-nearest neighbor graph, we extend 2-FNLS to search a set of promising neighbor solutions in NB4($x$). For each variable $x_{j_1}$ ($j_1 \in X$), we keep $j_2 \in (N \setminus X) \cap L'[j_1]$ that has the minimum value of $\Delta_{j_1, j_2}(x)$ in memory as $\pi(j_1)$. The extended 2-FNLS, called the 4-flip neighborhood search (4-FNLS) algorithm, then searches for an improved solution by flipping $x_{j_1} = 1 \rightarrow 0$, $x_{\pi(j_1)} = 0 \rightarrow 1$, $x_{j_2} = 1 \rightarrow 0$ and $x_{\pi(j_2)} = 0 \rightarrow 1$ for $j_1 \in X$. 

5
and \( j_3 \in X \cap L'[\pi(j_1)] \) satisfying \( j_1 \neq j_3 \) and \( \pi(j_1) \neq \pi(j_3) \), i.e., flipping the values of variables alternately along 4-paths or 4-cycles in the k-nearest neighbor graph. Let \( \Delta \tilde{z}_{j_1,j_2,j_3,j_4}(x) \) denote the increase in \( \tilde{z}(x) \) due to simultaneously flipping \( x_{j_3} = 1 \rightarrow 0 \), \( x_{j_2} = 0 \rightarrow 1 \), \( x_{j_3} = 1 \rightarrow 0 \) and \( x_{j_4} = 0 \rightarrow 1 \). 4-FNLS computes \( \Delta \tilde{z}_{j_1,j_2,j_3,j_4}(x) \) in \( O(|S_j|) \) time by applying the standard incremental evaluation alternately. 4-FNLS is formally described by replacing the part of the 2-FNLS algorithm after Step 2 with the following:

**Step 3’:** For each \( j_1 \in X \) in ascending order of \( \Delta \tilde{z}_{j_1}^+(x) \), if \( I_2^{(j_1)}(x) = \{ j_2 \in (N \setminus X) \cap L'[\pi(j_1)] \mid \Delta \tilde{z}_{j_1,j_2}(x) < 0 \} \neq \emptyset \) holds, choose \( j_2 \in I_2^{(j_1)}(x) \) that has the minimum value of \( \Delta \tilde{z}_{j_1,j_2}(x) \) 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 3’, return to Step 1.

**Step 4’:** For each \( j_1 \in X \) in ascending order of \( \Delta \tilde{z}_{j_1}^+(x) \), if \( I_4^{(j_1)}(x) = \{ j_3 \in X \cap L'[\pi(j_1)] \mid j_3 \neq j_1, \pi(j_3) \neq \pi(j_1), \Delta \tilde{z}_{j_1,j_3}(x) < 0 \} \neq \emptyset \) holds, choose \( j_3 \in I_4^{(j_1)}(x) \) that has the minimum value of \( \Delta \tilde{z}_{j_1,j_3}(x) \) and set \( x_{j_1} \leftarrow 0 \), \( x_{\pi(j_1)} \leftarrow 1 \), \( x_{j_3} \leftarrow 0 \) and \( x_{\pi(j_3)} \leftarrow 1 \). If the current solution \( x \) has been updated at least once in Step 4’, return to Step 1, otherwise output \( x \) and exit.

Although a similar approach has been developed in local search algorithms for the Euclidean traveling salesman problem (TSP) in which a sorted list containing only the k-nearest neighbors is stored for each city by using a geometric data structure called the k-dimensional tree [26], it is not suitable for finding the k-nearest neighbors efficiently in high-dimensional spaces. We thus extend it for application to the high-dimensional column vectors \( a_j \in \{0, 1\}^m \) (\( j \in N \)) of BIPs by using a lazy construction algorithm for the neighbor list.

### 4 Adaptive control of penalty weights

In our algorithm, solutions are evaluated by the alternative evaluation function \( \tilde{z}(x) \) in which the fixed penalty weight vectors \( w^+, w^- \) in the original evaluation function \( z(x) \) has been replaced by \( \widetilde{w}^+, \widetilde{w}^- \), and the values of \( \widetilde{w}^+_i \) (\( i \in M_L \cup M_E \)), \( \widetilde{w}^-_i \) (\( i \in M_G \cup M_E \)) are adaptively controlled in the search.

It is often reported that a single application of LS tends to stop at a locally optimal solution of insufficient quality when large penalty weights are used. This is because it is often unavoidable to temporarily increase the values of some violations \( y_i^+ \) and \( y_i^- \) in order to reach an even better solution from a good solution through a sequence of neighborhood operations, and large penalty weights thus prevent LS from moving between such solutions. To overcome this, we incorporate an adaptive adjustment mechanism for determining appropriate values of penalty weights \( \tilde{w}^+_i \) (\( i \in M_L \cup M_E \)), \( \tilde{w}^-_i \) (\( i \in M_G \cup M_E \)) [32] [44] [38]. That is, LS is applied iteratively while updating the values of the penalty weights \( \tilde{w}^+_i \) (\( i \in M_L \cup M_E \)), \( \tilde{w}^-_i \) (\( i \in M_G \cup M_E \)) after each call to LS. We call this sequence of calls to LS the weighting local search (WLS) according to [34] [36]. This strategy is also referred as the breakout algorithm [31] and the dynamic local search [25] in the literature.

Let \( x \) denote the solution at which the previous local search stops. We assume that the original penalty weights \( w^+_i \) (\( i \in M_L \cup M_E \)), \( w^-_i \) (\( i \in M_G \cup M_E \)) are sufficiently large. WLS resumes LS from \( x \) after updating the penalty weight vectors \( \tilde{w}^+, \tilde{w}^- \). Starting from the original penalty weight vectors \( (\tilde{w}^+, \tilde{w}^-) \leftarrow (w^+, w^-) \), the penalty weight vectors \( \tilde{w}^+, \tilde{w}^- \) are updated as follows. Let \( x^* \) denote the best solution obtained so far with respect to the original evaluation function \( z(x) \). If \( \tilde{z}(x) \geq z(x^*) \) holds, WLS uniformly decreases the penalty
weights by $(\tilde{\mathbf{w}}^+, \tilde{\mathbf{w}}^-) \leftarrow \beta(\mathbf{w}^+, \mathbf{w}^-)$, where $0 < \beta < 1$ is a program parameter that is adaptively computed so that the new value of $\Delta z^*_j(x)$ becomes negative for 10% of variables $x_j$ ($j \in X$). Otherwise, WLS increases the penalty weights by

$$
\tilde{w}^+_i \leftarrow \tilde{w}^+_i + \frac{z(x^*) - \tilde{z}(x)}{\sum_{l \in M}(y_l^2 + y_l^*)} y_i^+, \quad i \in M_L \cup M_E,
$$

$$
\tilde{w}^-_i \leftarrow \tilde{w}^-_i + \frac{z(x^*) - \tilde{z}(x)}{\sum_{l \in M}(y_l^2 + y_l^*)} y_i^-, \quad i \in M_G \cup M_E.
$$

(4)

WLS iteratively applies LS, updating the penalty weight vectors $\mathbf{w}^+$, $\mathbf{w}^-$ after each call to LS until the time limit is reached. Note that we modify 4-FNLS to evaluate solutions with both $\tilde{z}(x)$ and $z(x)$, and update the best solution $x^*$ with respect to the original objective function $z(x)$ whenever an improved solution is found. WLS is formally described as follows. Note that we set the initial solution to $x = 0$ in practice.

Algorithm WLS($x$)

**Input:** An initial solution $x$.

**Output:** The best solution $x^*$ with respect to $z(x)$.

**Step 1:** Set $x^* \leftarrow x$, $\bar{x} \leftarrow x$ and $(\bar{\mathbf{w}}^+, \bar{\mathbf{w}}^-) \leftarrow (\mathbf{w}^+, \mathbf{w}^-)$.

**Step 2:** Apply 4-FNLS($\bar{x}, \bar{\mathbf{w}}^+, \bar{\mathbf{w}}^-$) to obtain an improved solution $\bar{x}'$. Let $x'$ be the best solution with respect to the original evaluation function $z(x)$ obtained during the call to 4-FNLS($\bar{x}, \bar{\mathbf{w}}^+, \bar{\mathbf{w}}^-$). Set $\bar{x} \leftarrow \bar{x}'$.

**Step 3:** If $z(x') < z(x^*)$ holds, then set $x^* \leftarrow x'$. If the time limit is reached, output $x^*$ and halt.

**Step 4:** If $\tilde{z}(\bar{x}) \geq z(x^*)$ holds, then uniformly decrease the penalty weights by $(\tilde{\mathbf{w}}^+, \tilde{\mathbf{w}}^-) \leftarrow \beta(\mathbf{w}^+, \mathbf{w}^-)$, otherwise, increase the penalty weight vectors $(\tilde{\mathbf{w}}^+, \tilde{\mathbf{w}}^-)$ by (4) in practice. Return to Step 2.

5 Computational results

We report computational results of our algorithm for the SCP instances from [8, 38] and the SPP instances from [10, 20, 27]. Tables 1 and 2 summarize the information about the original and pre-solved instances. The first column shows the name of the group (or the instance), and the numbers in parentheses show the number of instances in the group. The second column “$z_{\text{LP}}$” shows the optimal values of the LP relaxation problems. The third column “$z_{\text{best}}$” shows the best upper bounds among all algorithms and the settings in this paper. The fourth and sixth columns “#cst.” show the number of constraints, and the fifth and seventh columns “#vars.” show the number of variables. Since several preprocessing techniques that often reduce the size of instances by removing redundant rows and columns are known [10], all algorithms are tested on the pre-solved instances. The instances marked “*” are hard instances that cannot be solved optimally within at least 1 h by the latest mixed integer programming (MIP) solvers.

We first compare our algorithm with three of the latest MIP solvers called CPLEX12.6, Gurobi5.6.3, and SCIP3.1 [1], and a local search solver called LocalSolver3.1 [9]. LocalSolver3.1 is not the latest version, but it performs better than the latest version (LocalSolver4.5) for SCP and SPP instances. We also compare our algorithm with a 3-flip local search algorithm developed by [44] (denoted by YKI) for SCP instances. All algorithms are tested on a MacBook.
Table 1: Benchmark instances for SCP

| instance   | \( z_{LP} \) | \( z_{best} \) | \( \#\text{cst.} \) | \( \#\text{var.} \) | \( \#\text{cst.} \) | \( \#\text{var.} \) | time limit |
|------------|-------------|--------------|----------------|-------------|----------------|-------------|------------|
| \( \star \)G.1–5 (5) | 149.48      | 166.4        | 1000.0         | 1000.0      | 1000.0        | 1000.0      | 600 s      |
| \( \star \)H.1–5 (5) | 45.67       | 59.6         | 1000.0         | 1000.0      | 1000.0        | 1000.0      | 600 s      |
| \( \star \)I.1–5 (5) | 138.97      | 158.0        | 1000.0         | 5000.0      | 1000.0        | 49981.0     | 1200 s     |
| \( \star \)J.1–5 (5) | 104.78      | 129.0        | 1000.0         | 10000.0     | 2000.0        | 99944.8     | 1200 s     |
| \( \star \)K.1–5 (5) | 276.67      | 313.2        | 2000.0         | 10000.0     | 2000.0        | 99971.0     | 1800 s     |
| \( \star \)L.1–5 (5) | 209.34      | 258.0        | 2000.0         | 20000.0     | 2000.0        | 199927.6    | 1800 s     |
| \( \star \)M.1–5 (5) | 415.78      | 549.8        | 5000.0         | 50000.0     | 5000.0        | 499888.0    | 3600 s     |
| RAIL507  | 348.93      | 503.8        | 10000.0        | 100000.0    | 5000.0        | 999993.2    | 3600 s     |
| RAIL516  | 172.15      | 174          | 507            | 63009       | 440           | 20700       | 600 s      |
| RAIL516  | 182.00      | 182          | 516            | 47311       | 403           | 37832       | 600 s      |
| RAIL582  | 209.71      | 211          | 582            | 55515       | 544           | 27427       | 600 s      |
| RAIL2536 | 688.40      | 689          | 2536           | 1081841     | 2001          | 480597      | 3600 s     |
| RAIL2586 | 935.92      | 947          | 2586           | 920683      | 2239          | 408724      | 3600 s     |
| RAIL4284 | 1054.05     | 1064         | 4284           | 1092610     | 3633          | 607884      | 3600 s     |
| RAIL4872 | 1509.64     | 1530         | 4872           | 968672      | 4207          | 482500      | 3600 s     |

Table 2: Benchmark instances for SPP

| instance   | \( z_{LP} \) | \( z_{best} \) | \( \#\text{cst.} \) | \( \#\text{var.} \) | \( \#\text{cst.} \) | \( \#\text{var.} \) | time limit |
|------------|-------------|--------------|----------------|-------------|----------------|-------------|------------|
| aa01–06   (6) | 40372.75    | 40588.83     | 675.3          | 7587.3      | 478.7         | 6092.7      | 600 s      |
| us01–04   (4) | 9749.44     | 9798.25      | 121.3          | 295085.0    | 65.5          | 85772.5     | 600 s      |
| t0415–0421 (7) | 5199083.74 | 5453475.71   | 1479.3         | 7304.3      | 820.7         | 2617.4      | 600 s      |
| \( \star \)t1716–1722 (7) | 121445.76 | 158739.86    | 475.7          | 58981.3     | 475.7         | 13193.6     | 3600 s     |
| v0415–0421 (7) | 2385764.17 | 2393303.71   | 1479.3         | 30341.6     | 263.9         | 7277.0      | 600 s      |
| v1616–1622 (7) | 1021288.76 | 1025552.43   | 1375.7         | 83986.7     | 1171.9        | 51136.7     | 600 s      |
| \( \star \)ds  | 57.23       | 187.47       | 656            | 67732       | 656           | 67730       | 3600 s     |
| \( \star \)ds-big | 86.82       | 731.69       | 1042           | 174997      | 1042          | 173026      | 3600 s     |
| \( \star \)ivu06-big | 135.43       | 166.02       | 1177           | 2277736     | 1177          | 2197774     | 3600 s     |
| \( \star \)ivu59 | 884.46      | 1878.83      | 3436           | 2569996     | 3413          | 2565083     | 3600 s     |

Pro laptop computer with a 2.7 GHz Intel Core i7 processor, and are run on a single thread with time limits as shown in Tables 1 and 2.

Tables 3 and 4 show the relative gap (%) \( \frac{(x) - z_{best}}{z(x)} \times 100 \) of the best feasible solutions achieved by the algorithms under the original (hard) constraints. The numbers in parentheses show the number of instances for which the algorithm obtained at least one feasible solution within the time limit.

We first observe that our algorithm achieves good upper bounds comparable to those of the 3-flip neighborhood local search [44] for many SCP instances. We note that the 3-flip neighborhood local search algorithm introduces a heuristic variable fixing technique based on Lagrangian relaxation that substantially reduces the number of variables to be considered to 1.05% from the original SCP instances on average. We next observe that our algorithm achieves best upper bounds in 17 out of 42 instances for all SPP instances, including 7 out of 11 instances for hard instances in particular. We note that local search algorithms and MIP solvers are quite different in character. Local search algorithms do not guarantee optimality because they typically search only a portion of the solution space. MIP solvers, however, examine every possible solution, at least implicitly, in order to guarantee optimality. Hence, it is inherently difficult to find optimal solutions by local search algorithms even in the case of small instances.
Table 3: Comparison of the best gap of the proposed algorithm to those of the latest solvers for SCP instances

| instance       | CPLEX12.6 | Gurobi5.6.3 | SCIP3.1 | LocalSolver3.1 | YKI | proposed |
|----------------|-----------|-------------|---------|-----------------|-----|----------|
| ★G.1–5 (5)    | 0.37%     | 0.49%       | 0.24%   | 45.80%          | 0.00%| 0.00%    |
| ★H.1–5 (5)    | 1.92%     | 2.28%       | 1.93%   | 61.54%          | 0.00%| 0.00%    |
| ★L.1–5 (5)    | 2.81%     | 2.72%       | 1.85%   | 41.38%          | 0.00%| 0.50%    |
| ★J.1–5 (5)    | 8.37%     | 4.30%       | 3.59%   | 58.40%          | 0.00%| 1.53%    |
| ★K.1–5 (5)    | 4.77%     | 4.38%       | 2.55%   | 51.22%          | 0.00%| 1.26%    |
| ★L.1–5 (5)    | 9.57%     | 8.44%       | 3.52%   | 57.79%          | 0.00%| 2.05%    |
| ★M.1–5 (5)    | 18.43%    | 10.16%      | 30.71%  | 71.08%          | 0.00%| 2.65%    |
| ★N.1–5 (5)    | 33.13%    | 12.49%      | 42.32%  | 75.63%          | 0.00%| 5.47%    |
| RAIL507       | 0.00%     | 0.00%       | 0.00%   | 5.43%           | 0.00%| 0.00%    |
| RAIL516       | 0.00%     | 0.00%       | 0.00%   | 3.19%           | 0.00%| 0.00%    |
| RAIL582       | 0.00%     | 0.00%       | 0.00%   | 5.80%           | 0.00%| 0.00%    |
| RAIL2536      | 0.00%     | 0.00%       | 0.86%   | 3.50%           | 0.29%| 0.72%    |
| ★RAIL2586     | 2.27%     | 2.17%       | 2.27%   | 5.39%           | 0.00%| 1.56%    |
| ★RAIL4284     | 5.34%     | 1.57%       | 30.55%  | 6.50%           | 0.00%| 2.12%    |
| ★RAIL4872     | 1.73%     | 1.73%       | 2.67%   | 5.61%           | 0.00%| 1.80%    |
| avg. (47)     | 8.64%     | 4.92%       | 10.00%  | 49.99%          | 0.01%| 1.56%    |

Table 4: Comparison of the best gap of the proposed algorithm to those of the latest solvers for SPP instances

| instance       | CPLEX12.6 | Gurobi5.6.3 | SCIP3.1 | LocalSolver3.1 | proposed |
|----------------|-----------|-------------|---------|-----------------|----------|
| aa01–06 (6)    | 0.00% (6) | 0.00% (6)   | 0.00% (6) | 13.89% (1) | 1.60% (6) |
| us01–04 (4)    | 0.00% (4) | 0.00% (4)   | 0.00% (3) | 11.26% (2) | 0.04% (4) |
| t0415–0421 (7) | 0.66% (7) | 0.60% (7)   | 1.61% (6) | — (0)       | 0.92% (6) |
| *t1716–1722 (7)| 7.63% (7) | 15.94% (7)  | 2.76% (7) | 36.09% (1) | 1.70% (7) |
| v0415–0421 (7) | 0.00% (7) | 0.00% (7)   | 0.00% (7) | 0.05% (7)  | 0.00% (7) |
| v1616–1622 (7) | 0.00% (7) | 0.00% (7)   | 0.00% (7) | 4.60% (7)  | 0.09% (7) |
| ★ds            | 8.86%     | 55.61%      | 40.53%  | 85.17%         | 0.00%    |
| ★ds-big        | 62.16%    | 24.03%      | 72.01%  | 92.69%         | 0.00%    |
| ★vivu06-big    | 20.86%    | 0.68%       | 17.90%  | 52.54%         | 0.00%    |
| ★vivu59        | 28.50%    | 4.36%       | 37.84%  | 48.95%         | 0.00%    |
| avg. (42)      | 4.25% (42)| 4.77% (42)  | 4.93% (40)| 17.47% (22) | 0.69% (41) |

while MIP solvers find optimal solutions quickly for small instances and/or those having a small gap between the lower and upper bounds of optimal values. In view of this, our algorithm achieves sufficiently good upper bounds compared to the other algorithms on the benchmark instances, particularly for the hard SPP instances.

Tables 5 and 6 show the completion rate of the neighbor list by rows, i.e., the proportion of generated rows to all rows in the neighbor list. We observe that our algorithm achieves good performance while generating only a small part of the neighbor list for the large-scale instances.

Tables 7 and 8 show the relative gap of the best feasible solutions achieved by our algorithm for different settings. The second column “no-list” shows the results of our algorithm without the neighbor list, and the third column “no-inc” shows the results of our algorithm without the improved incremental evaluation (i.e., only applying the standard incremental evaluation). The fourth column “2-FNLS” shows the results of our algorithm without the 4-flip neighborhood search (i.e., only applying 2-FNLS). Tables 9 and 10 also show the computational efficiency of our algorithm for different settings with respect to the number of calls to 4-FNLS (and 2-FNLS in the fourth column “2-FNLS”), where the values are normalized so that values of the proposed
Table 5: Completion rate of the neighbor list by rows for SCP instances

| instance | 1 min | 10 min | 20 min | 30 min | 1 h  |
|----------|-------|--------|--------|--------|------|
| ⋆G.1–5 (5) | 3.58% | 3.95% | —      | —      | —    |
| ⋆H.1–5 (5) | 2.12% | 2.43% | —      | —      | —    |
| ⋆I.1–5 (5) | 1.39% | 1.63% | 1.71%  | —      | —    |
| ⋆J.1–5 (5) | 0.82% | 1.10% | 1.16%  | —      | —    |
| ⋆K.1–5 (5) | 1.20% | 1.49% | 1.57%  | 1.61%  | —    |
| ⋆L.1–5 (5) | 0.57% | 0.98% | 1.06%  | 1.10%  | —    |
| ⋆M.1–5 (5) | 0.20% | 0.58% | 0.73%  | 0.81%  | 0.93%|
| ⋆N.1–5 (5) | 0.01% | 0.21% | 0.29%  | 0.35%  | 0.49%|
| RAIL507 | 13.41% | 22.02% | —      | —      | —    |
| RAIL516 | 4.73% | 8.94% | —      | —      | —    |
| RAIL582 | 8.19% | 10.66% | —      | —      | —    |
| RAIL2536 | 0.30% | 1.36% | 1.87%  | 2.17%  | 2.73%|
| ⋆RAIL2586 | 0.44% | 1.67% | 2.19%  | 2.54%  | 3.28%|
| ⋆RAIL4284 | 0.20% | 1.01% | 1.43%  | 1.73%  | 2.29%|
| ⋆RAIL4872 | 0.37% | 1.56% | 2.11%  | 2.46%  | 3.10%|

Table 6: Completion rate of the neighbor list by rows for SPP instances

| instance | 1 min | 10 min | 30 min | 1 h  |
|----------|-------|--------|--------|------|
| aa01–06 (6) | 40.47% | 49.17% | —      | —    |
| us01–04 (4) | 3.93% | 5.17% | —      | —    |
| t0415–0421 (7) | 83.47% | 90.00% | —      | —    |
| ⋆t1716–1722 (7) | 61.00% | 94.38% | 97.12% | 97.98%|
| v0415–0421 (7) | 31.44% | 31.64% | —      | —    |
| v1616–1622 (7) | 6.55% | 7.42% | —      | —    |
| ⋆ds | 2.29% | 12.63% | 27.11% | 40.05%|
| ⋆ds-big | 0.21% | 2.06% | 4.96%  | 8.10% |
| ⋆ivu06-big | 0.01% | 0.07% | 0.23%  | 0.45% |
| ⋆ivus59 | 0.01% | 0.05% | 0.11%  | 0.16% |

From the results, we can see that the overall computational efficiency of our algorithm was improved 16.90-fold and 17.26-fold on average for SCP and SPP instances, respectively in comparison with the naive algorithm that has neither the neighbor list nor the improved incremental evaluation (i.e., only applying the standard incremental evaluation), and our algorithm attains good performance even when the size of the neighbor list is considerably small. We also observe that the 4-flip neighborhood search substantially improves the performance of our algorithm, even though there are fewer calls to 4-FNLS compared to 2-FNLS.

6 Conclusion

We present a data mining approach for reducing the search space of local search algorithms for a class of BIPs including SCP and SPP. In this approach, we construct a $k$-nearest neighbor graph by extracting variable associations from the instance to be solved in order to identify promising pairs of flipping variables in the 2-flip neighborhood. We also develop a 4-flip neighborhood local search algorithm that flips four variables alternately along 4-paths or 4-cycles in the $k$-nearest neighbor graph. We incorporate an efficient incremental evaluation of solutions and an adaptive control of penalty weights into the 4-flip neighborhood local search algorithm. Computational
Table 7: Comparison of the best gap of variations of the proposed algorithm for SCP instances

| instance   | no-list | no-inc | 2-FNLS | proposed |
|------------|---------|--------|--------|----------|
| *G.1–5 (5) | 0.00%   | 0.12%  | 0.00%  | 0.00%    |
| *H.1–5 (5) | 0.31%   | 0.31%  | 0.31%  | 0.00%    |
| *I.1–5 (5) | 1.24%   | 0.86%  | 0.50%  | 0.50%    |
| *J.1–5 (5) | 2.42%   | 1.67%  | 1.68%  | 1.53%    |
| *K.1–5 (5) | 2.12%   | 1.69%  | 1.32%  | 1.26%    |
| *L.1–5 (5) | 3.44%   | 3.51%  | 2.35%  | 2.05%    |
| *M.1–5 (5) | 10.97%  | 8.33%  | 2.79%  | 2.65%    |
| *N.1–5 (5) | 19.11%  | 22.06% | 4.76%  | 5.47%    |
| RAIL507    | 0.00%   | 0.57%  | 0.00%  | 0.00%    |
| RAIL516    | 0.00%   | 0.00%  | 0.00%  | 0.00%    |
| RAIL582    | 0.47%   | 0.47%  | 0.47%  | 0.00%    |
| RAIL2536   | 2.68%   | 2.27%  | 1.29%  | 0.72%    |
| *RAIL2586  | 2.57%   | 2.87%  | 2.27%  | 1.56%    |
| *RAIL4284  | 5.42%   | 5.17%  | 2.74%  | 2.12%    |
| *RAIL4872  | 4.43%   | 3.47%  | 2.36%  | 1.80%    |
| avg. (47)  | 4.55%   | 4.42%  | 1.65%  | 1.56%    |

Table 8: Comparison of the best gap of variations of the proposed algorithm for SPP instances

| instance   | no-list | no-inc | 2-FNLS | proposed |
|------------|---------|--------|--------|----------|
| aa01–06 (6)| 2.33%   | 2.26%  | 2.07%  | 1.60%    |
| us01–04 (4)| 0.04%   | 1.16%  | 0.63%  | 0.04%    |
| t0415–0421 (7)| 1.46% | 1.30% | 0.29% | 0.92% |
| *t1716–1722 (7)| 4.73% | 3.58% | 4.98% | 1.70% |
| v0415–0421 (7)| 0.00% | 0.00% | 0.00% | 0.00% |
| v1616–1622 (7)| 0.62% | 0.17% | 0.09% | 0.09% |
| *ds       | 36.03%  | 33.80% | 24.13% | 0.00%    |
| *ds-big   | 29.11%  | 0.00%  | 40.75% | 0.00%    |
| *ivu06-big| 5.31%   | 3.83%  | 2.25%  | 0.00%    |
| *ivu59    | 15.75%  | 11.39% | 16.01% | 0.00%    |
| avg. (42) | 3.63%   | 2.47%  | 2.33%  | 0.69%    |

comparison with the latest solvers shows that our algorithm achieves sufficiently good upper bounds for SCP and SPP instances, particularly for hard SPP instances.

We expect that data mining approaches could also be beneficial for efficiently solving other large-scale combinatorial optimization problems, particularly for hard instances that have large gaps between the lower and upper bounds of the optimal values.

References

[1] Achterberg, T. (2009). SCIP: Solving constraint integer programs. *Mathematical Programming Computation*, 1, 1–41.

[2] Agarwal, Y., Mathur, K., & Salkin, H. M. (1989). A set-partitioning-based exact algorithm for the vehicle routing problem. *Networks*, 19, 731–749.

[3] Atamtürk, A., Nemhauser, G. L., & Savelsbergh, M. W. P. (1995). A combined Lagrangian, linear programming, and implication heuristic for large-scale set partitioning problems. *Journal of Heuristics*, 1, 247–259.
Table 9: Comparison of the computational efficiency of variations of the proposed algorithm for SCP instances

| instance   | no-list | no-inc | 2-FNLS | proposed |
|------------|---------|--------|--------|----------|
| C.1–5 (5)  | 0.51    | 0.36   | 1.92   | 1.00     |
| H.1–5 (5)  | 0.83    | 0.35   | 1.76   | 1.00     |
| L.1–5 (5)  | 0.39    | 0.34   | 2.33   | 1.00     |
| J.1–5 (5)  | 0.47    | 0.43   | 2.44   | 1.00     |
| K.1–5 (5)  | 0.25    | 0.31   | 1.98   | 1.00     |
| L.1–5 (5)  | 0.32    | 0.34   | 2.31   | 1.00     |
| M.1–5 (5)  | 0.19    | 0.22   | 2.65   | 1.00     |
| N.1–5 (5)  | 0.27    | 0.23   | 3.20   | 1.00     |
| RAIL507    | 0.19    | 0.27   | 1.61   | 1.00     |
| RAIL516    | 0.16    | 0.27   | 1.37   | 1.00     |
| RAIL582    | 0.20    | 0.27   | 1.46   | 1.00     |
| RAIL2536   | 0.13    | 0.11   | 1.26   | 1.00     |
| RAIL2586   | 0.14    | 0.13   | 1.78   | 1.00     |
| RAIL4284   | 0.11    | 0.12   | 1.58   | 1.00     |
| RAIL4872   | 0.08    | 0.10   | 1.78   | 1.00     |
| avg. (47)  | 0.37    | 0.30   | 2.21   | 1.00     |

Table 10: Comparison of computational efficiency of variations of the proposed algorithm for SPP instances

| instance   | no-list | no-inc | 2-FNLS | proposed |
|------------|---------|--------|--------|----------|
| aa01–06 (6)| 0.18    | 0.37   | 1.66   | 1.00     |
| us01–04 (4)| 0.36    | 0.45   | 1.07   | 1.00     |
| t0415–0421 (7)| 0.17 | 0.33   | 2.71   | 1.00     |
| t1716–1722 (7)| 0.18 | 0.35   | 1.62   | 1.00     |
| v0415–0421 (7)| 0.20 | 0.42   | 1.53   | 1.00     |
| v1616–1622 (7)| 0.15 | 0.41   | 1.58   | 1.00     |
| ds         | 0.27    | 0.30   | 1.24   | 1.00     |
| ds-big     | 0.26    | 0.31   | 1.33   | 1.00     |
| ivu06-big  | 0.22    | 0.29   | 1.22   | 1.00     |
| ivu59      | 0.32    | 0.48   | 1.42   | 1.00     |
| avg. (42)  | 0.20    | 0.38   | 1.70   | 1.00     |

[4] Balas, E., & Padberg, M. W. (1976). Set partitioning: A survey. *SIAM Review*, 18, 710–760.

[5] Barahona, F. & Anbil, R. (2000). The volume algorithm: Producing primal solutions with a subgradient method. *Mathematical Programming*, A87, 385–399.

[6] Barnhart, C., Johnson, E. L., Nemhauser, G. L., Savelsbergh, M. W. P., & Vance, P. H. (1998). Branch-and-price: Column generation for solving huge integer programs. *Operations Research*, 46, 316–329.

[7] Bastert, O., Hummel, B. & de Vries, S. (2010). A generalized Wedelin heuristic for integer programming. *INFORMS Journal on Computing*, 22, 93–107.

[8] Beasley, J. E. (1990). OR-Library: Distributing test problems by electronic mail. *Journal of the Operational Research Society*, 41, 1069–1072.
[9] Benoist, T., Estellon, B., Gardi, F., Megel, R., & Nouioua, K. (2011). LocalSolver 1.x: A black-box local-search solver for 0-1 programming. *4OR — A Quarterly Journal of Operations Research*, 9, 299–316.

[10] Borndörfer, R. (1998). *Aspects of set packing, partitioning and covering*. Ph. D. Dissertation, Berlin: Technischen Universität.

[11] Boros, E., Hammer, P. L., Ibaraki, T., Kogan, A., Mayoraz, E., & Muchnik, I. (2000). An implementation of logical analysis of data. *IEEE Transactions on Knowledge and Data Engineering*, 12, 292–306.

[12] Boros, E., Ibaraki, T., Ichikawa, H., Nonobe, K., Uno, T., & Yagiura, M. (2005). Heuristic approaches to the capacitated square covering problem. *Pacific Journal of Optimization*, 1, 465–490.

[13] Boschetti, M. A., Mingozzi, A. & Ricciardelli, S. (2008). A dual ascent procedure for the set partitioning problem. *Discrete Optimization*, 5, 735–747.

[14] Bramel, J., & Simchi-Levi, D. (1997). On the effectiveness of set covering formulations for the vehicle routing problem with time windows. *Operations Research*, 45, 295–301.

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

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

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

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

[19] Ceria, S., Nobili, P., & Sassano, A. (1998). A Lagrangian-based heuristic for large-scale set covering problems. *Mathematical Programming*, 81, 215–288.

[20] Chu P. C., & Beasley, J. E. (1998). Constraint handling in genetic algorithms: The set partitioning problem. *Journal of Heuristics*, 11, 323–357.

[21] Farahani, R. Z., Asgari, N., Heidari, N., Hosseininia, M., & Goh, M. (2012). Covering problems in facility location: A review. *Computers & Industrial Engineering*, 62, 368–407.

[22] Hammer, P. L., & Bonates, T. O. (2006). Logical analysis of data — An overview: From combinatorial optimization to medical applications. *Annals of Operations Research*, 148, 203–225.

[23] Hashimoto, H., Ezaki, Y., Yagiura, M., Nonobe, K., Ibaraki, T., & Løkketangen, A. (2009). A set covering approach for the pickup and delivery problem with general constraints on each route. *Pacific Journal of Optimization*, 5, 183–200.

[24] Hoffman, K. L., & Padberg, A. (1993). Solving airline crew scheduling problems by branch-and-cut. *Management Science*, 39, 657–682.
[25] Hutter, F., Tompkins, D. A. D., & Hoos, H. H. (2002). Scaling and probabilistic smoothing: Efficient dynamic local search for SAT. *Proceedings of International Conference on Principles and Practice of Constraint Programming (CP)*, 233–248.

[26] Johnson, D. S., & McGeoch, L. A. (1997). The traveling salesman problem: A case study. In E. Aarts, & K. Lenstra (eds.), *Local Search in Combinatorial Optimization* (pp. 215–310). New Jersey: Princeton University Press.

[27] Koch, T., Achterberg, T., Andersen, E., Bastert, O., Berthold, T., Bixby, R. E., Danna, E., Gamrath, G., Gleixner, A. M., Heinz, S, Lodi, A., Mittelmann, H., Ralphi, T., Salvagnin, D., Steffy, D. E., & Wolter, K. (2011). MIPLIB2010: Mixed integer programming library version 5. *Mathematical Programming Computation*, 3, 103–163.

[28] Linderoth, J. T., Lee, E. K., & Savelbergh, M. W. P. (2001). A parallel, linear programming-based heuristic for large-scale set partitioning problems. *INFORMS Journal on Computing*, 13, 191–209.

[29] Michel, L. & Van Hentenryck, P. (2000). Localizer. *Constraints: An International Journal*, 5, 43–84.

[30] Mingozzi, A., Boschetti, M. A., Ricciardelli, S., & Bianco, L. (1999). A set partitioning approach to the crew scheduling problem. *Operations Research*, 47, 873–888.

[31] Morris, P. (1993). The breakout method for escaping from local minima. *Proceedings of National Conference on Artificial Intelligence (AAAI)*, 40–45.

[32] Nonobe, K., & Ibaraki, T. (2001). An improved tabu search method for the weighted constraint satisfaction problem. *INFOR*, 39, 131–151.

[33] Pesant, G., & Gendreau, M. (1999). A constraint programming framework for local search methods. *Journal of Heuristics*, 5, 255–279.

[34] Selman, B., & Kautz, H. (1993). Domain-independent extensions to GSAT: Solving large structured satisfiability problems. *Proceedings of International Conference on Artificial Intelligence (IJCAI)*, 290–295.

[35] Shaw, P., Backer, B. D., & Furnon, V. (2002). Improved local search for CP toolkits. *Annals of Operations Research*, 115, 31–50.

[36] Thornton, J. (2005). Clause weighting local search for SAT. *Journal of Automated Reasoning*, 35, 97–142.

[37] Umetani, S., & Yagiura, M. (2007). Relaxation heuristics for the set covering problem. *Journal of the Operations Research Society of Japan*, 50, 350–375.

[38] Umetani, S., Arakawa, M., & Yagiura, M. (2013). A heuristic algorithm for the set multiset cover problem with generalized upper bound constraints. *Proceedings of Learning and Intelligent Optimization Conference (LION)*, 75–80.

[39] Umetani, S. (2015). Exploiting variable associations to configure efficient local search in large-scale set partitioning problems. *Proceedings of AAAI Conference on Artificial Intelligence (AAAI)*, 1226–1232.

[40] Van Hentenryck, P., & Michel, L. (2005). *Constraint-Based Local Search*, Cambridge: The MIT Press.
The following formulas:

\[ \Delta \tilde{z}_{j_1,j_2}(x) \geq 0 \] holds if \( x_{j_1} = x_{j_2} \). First, we consider the case of \( x_{j_1} = x_{j_2} = 1 \). By the assumption of the lemma,

\[ \Delta \tilde{z}_j^*(x) = \sum_{i \in S_j \cap (M_L \cup M_E) \cap \{ i | s_i(x) > b_i \}} \tilde{w}_i^+ + \sum_{i \in S_j \cap (M_G \cup M_E) \cap \{ i | s_i(x) \leq b_i \}} \tilde{w}_i^- \geq 0 \] (5)

holds for both \( j = j_1 \) and \( j_2 \). We then have

\[ \Delta \tilde{z}_{j_1,j_2}(x) = \Delta \tilde{z}_{j_1}^+(x) + \Delta \tilde{z}_{j_2}^+(x) + \sum_{i \in S_{j_1} \cap S_{j_2} \cap (M_L \cup M_E) \cap M_+(x)} \tilde{w}_i^+ + \sum_{i \in S_{j_1} \cap S_{j_2} \cap (M_G \cup M_E) \cap M_+(x)} \tilde{w}_i^- \geq 0 \] (6)

where \( M_+(x) = \{ i \in M | s_i(x) = b_i + 1 \} \). Next, we consider the case of \( x_{j_1} = x_{j_2} = 0 \). By the assumption of the lemma,

\[ \Delta \tilde{z}_j^*(x) = \sum_{i \in S_j \cap (M_L \cup M_E) \cap \{ i | s_i(x) \geq b_i \}} \tilde{w}_i^+ - \sum_{i \in S_j \cap (M_G \cup M_E) \cap \{ i | s_i(x) < b_i \}} \tilde{w}_i^- \geq 0 \] (7)

holds for both \( j = j_1 \) and \( j_2 \). We then have

\[ \Delta \tilde{z}_{j_1,j_2}(x) = \Delta \tilde{z}_{j_1}^+(x) + \Delta \tilde{z}_{j_2}^+(x) + \sum_{i \in S_{j_1} \cap S_{j_2} \cap (M_L \cup M_E) \cap M_-(x)} \tilde{w}_i^+ + \sum_{i \in S_{j_1} \cap S_{j_2} \cap (M_G \cup M_E) \cap M_-(x)} \tilde{w}_i^- \geq 0 \] (8)

where \( M_-(x) = \{ i \in M | s_i(x) = b_i - 1 \} \).

### Appendix A Proof of Lemma [1]

We show that \( \Delta \tilde{z}_{j_1,j_2}(x) \geq 0 \) holds if \( x_{j_1} = x_{j_2} \). First, we consider the case of \( x_{j_1} = x_{j_2} = 1 \). By the assumption of the lemma,

\[ \Delta \tilde{z}_j^*(x) = \sum_{i \in S_j \cap (M_L \cup M_E) \cap \{ i | s_i(x) > b_i \}} \tilde{w}_i^+ + \sum_{i \in S_j \cap (M_G \cup M_E) \cap \{ i | s_i(x) \leq b_i \}} \tilde{w}_i^- \geq 0 \] (5)

holds for both \( j = j_1 \) and \( j_2 \). We then have

\[ \Delta \tilde{z}_{j_1,j_2}(x) = \Delta \tilde{z}_{j_1}^+(x) + \Delta \tilde{z}_{j_2}^+(x) + \sum_{i \in S_{j_1} \cap S_{j_2} \cap (M_L \cup M_E) \cap M_+(x)} \tilde{w}_i^+ + \sum_{i \in S_{j_1} \cap S_{j_2} \cap (M_G \cup M_E) \cap M_+(x)} \tilde{w}_i^- \geq 0 \] (6)

where \( M_+(x) = \{ i \in M | s_i(x) = b_i + 1 \} \). Next, we consider the case of \( x_{j_1} = x_{j_2} = 0 \). By the assumption of the lemma,

\[ \Delta \tilde{z}_j^*(x) = \sum_{i \in S_j \cap (M_L \cup M_E) \cap \{ i | s_i(x) \geq b_i \}} \tilde{w}_i^+ - \sum_{i \in S_j \cap (M_G \cup M_E) \cap \{ i | s_i(x) < b_i \}} \tilde{w}_i^- \geq 0 \] (7)

holds for both \( j = j_1 \) and \( j_2 \). We then have

\[ \Delta \tilde{z}_{j_1,j_2}(x) = \Delta \tilde{z}_{j_1}^+(x) + \Delta \tilde{z}_{j_2}^+(x) + \sum_{i \in S_{j_1} \cap S_{j_2} \cap (M_L \cup M_E) \cap M_-(x)} \tilde{w}_i^+ + \sum_{i \in S_{j_1} \cap S_{j_2} \cap (M_G \cup M_E) \cap M_-(x)} \tilde{w}_i^- \geq 0 \] (8)

where \( M_-(x) = \{ i \in M | s_i(x) = b_i - 1 \} \).

### Appendix B Efficient incremental evaluation

We first consider a standard incremental evaluation of \( \Delta \tilde{z}_j^+(x) \) and \( \Delta \tilde{z}_j^-(x) \) in \( O(|S_j|) \) time using the following formulas:

\[
\Delta \tilde{z}_j^+(x) = c_j + \Delta \tilde{p}_j^+(x) + \Delta \tilde{q}_j^+(x), \\
\Delta \tilde{p}_j^+(x) = \sum_{i \in S_j \cap (M_L \cup M_E)} \tilde{w}_i^+ ([s_i(x) + 1] - b_i_+ - |s_i(x) - b_i|_+), \\
\Delta \tilde{q}_j^+(x) = \sum_{i \in S_j \cap (M_L \cup M_E)} \tilde{w}_i^- ([b_i - (s_i(x) + 1)]_+ - |b_i - s_i(x)|_+),
\]

(9)
Similarly, when the current solution \( x \) moves to \( x' \) by flipping \( x_j = 0 \) and \( x_j = 1 \), 2-FNLS first updates \( s_i(\mathbf{x}) \) \( (i \in S) \) in \( O(|S|) \) time by \( s_i(x') \leftarrow s_i(x) + 1 \) and \( s_i(x') \leftarrow s_i(x) - 1 \) when the current solution \( \mathbf{x} \) moves to \( \mathbf{x}' \) by flipping \( x_j = 0 \) and \( x_j = 1 \), respectively.

We further develop an improved incremental evaluation of \( \Delta z^+_j(\mathbf{x}) \) and \( \Delta z^-_j(\mathbf{x}) \) in \( O(1) \) time by directly keeping \( \Delta \tilde{p}^+_j(\mathbf{x}), \Delta \tilde{q}^+_j(\mathbf{x}) \) \( (j \in N \setminus X) \) and \( \Delta \tilde{p}^-_j(\mathbf{x}), \Delta \tilde{q}^-_j(\mathbf{x}) \) \( (j \in X) \) in memory. When the current solution \( \mathbf{x} \) moves to \( \mathbf{x}' \) by flipping \( x_j = 0 \) to 1, 2-FNLS first updates \( s_i(\mathbf{x}) \) \( (i \in S) \) in \( O(|S|) \) time by \( s_i(x') \leftarrow s_i(x) + 1 \), and then updates \( \Delta \tilde{p}^+_k(\mathbf{x}), \Delta \tilde{q}^+_k(\mathbf{x}), \Delta \tilde{p}^-_k(\mathbf{x}), \Delta \tilde{q}^-_k(\mathbf{x}) \) \( (k \in N_i, i \in S) \) in \( O(\sum_{i \in S} |N_i|) \) time using the following formulas:

\[
\begin{align*}
\Delta \tilde{p}^+_k(\mathbf{x}') &\leftarrow \Delta \tilde{p}^+_k(\mathbf{x}) + \sum_{i \in S_j \cap S_k \cap (M_L \cup M_E)} \tilde{w}^+_i (\Delta y^+_i(\mathbf{x}') - \Delta y^+_i(\mathbf{x})) , \\
\Delta \tilde{q}^+_k(\mathbf{x}') &\leftarrow \Delta \tilde{q}^+_k(\mathbf{x}) + \sum_{i \in S_j \cap S_k \cap (M_L \cup M_E)} \tilde{w}^-_i (\Delta y^-_i(\mathbf{x}') - \Delta y^-_i(\mathbf{x})) , \\
\Delta \tilde{p}^-_k(\mathbf{x}') &\leftarrow \Delta \tilde{p}^-_k(\mathbf{x}) + \sum_{i \in S_j \cap S_k \cap (M_L \cup M_E)} \tilde{w}^+_i (\Delta y^+_i(\mathbf{x}') - \Delta y^+_i(\mathbf{x})) , \\
\Delta \tilde{q}^-_k(\mathbf{x}') &\leftarrow \Delta \tilde{q}^-_k(\mathbf{x}) + \sum_{i \in S_j \cap S_k \cap (M_L \cup M_E)} \tilde{w}^-_i (\Delta y^-_i(\mathbf{x}') - \Delta y^-_i(\mathbf{x})) ,
\end{align*}
\]

where

\[
\begin{align*}
\Delta y^+_i(\mathbf{x}') &= |(s_i(\mathbf{x}') + 1) - b_i|_+ - |s_i(\mathbf{x}') - b_i|_+ , \\
\Delta y^-_i(\mathbf{x}) &= |s_i(\mathbf{x}) - b_i|_+ - |s_i(\mathbf{x}) - b_i|_+ , \\
\Delta y^+_i(\mathbf{x}) &= |s_i(\mathbf{x}) - b_i|_+ - |s_i(\mathbf{x}) - b_i|_+ , \\
\Delta y^-_i(\mathbf{x'}) &= |b_i - (s_i(\mathbf{x}) + 1)|_+ - |b_i - s_i(\mathbf{x})|_+ , \\
\Delta y^-_i(\mathbf{x}) &= |b_i - s_i(\mathbf{x})|_+ - |b_i - s_i(\mathbf{x})|_+ .
\end{align*}
\]

Similarly, when the current solution \( \mathbf{x} \) moves to \( \mathbf{x}' \) by flipping \( x_j = 1 \) to 0, 2-FNLS first updates \( s_i(\mathbf{x}) \) \( (i \in S) \) in \( O(|S|) \) time, and then updates \( \Delta \tilde{p}^-_k(\mathbf{x}), \Delta \tilde{q}^-_k(\mathbf{x}), \Delta \tilde{p}^+_k(\mathbf{x}), \Delta \tilde{q}^+_k(\mathbf{x}) \) \( (k \in N_i, i \in S) \) in \( O(\sum_{i \in S} |N_i|) \) time. We note that the computation time for updating the auxiliary data has little effect on the total computation time of 2-FNLS because the number of solutions actually visited is much less than the number of neighbor solutions evaluated in most cases.