Learning-based Multi-Start Iterated Local Search for the Profit Maximization Set Covering Problem

Wen Sun [a,b], Wenlong Li [a], Jin-Kao Hao [c,*], Qinghua Wu [d]

[a] *School of Cyber Science and Engineering, Southeast University, 2 Road Southeast University, 211189 Nanjing, China*

[b] *Frontiers Science Center for Mobile Information Communication and Security, Southeast University, 2 Road Southeast University, 211189 Nanjing, China*

[c] *LERIA, Université d'Angers, 2 Boulevard Lavoisier, 49045 Angers, France*

[d] *School of Management, Huazhong University of Science and Technology, 1037 Road Luoyu, 430074 Wuhan, China*

Information Sciences, https://doi.org/10.1016/j.ins.2023.119404

Abstract

The profit maximization set covering problem is a general model able to formulate practical problems including in particular an application in the mining industry. As a variant of the partial set covering problem, the studied problem is to select some subsets of elements to maximize the difference of the total gain of the covered elements subtracting the costs of the chosen subsets and their associated groups. We investigate for the first time a learning-based multi-start iterated local search algorithm for solving the problem. The proposed algorithm combines a multi-restart mechanism to enhance robustness, an intensification-driven local search to perform intensive solution examination, a learning-driven initialization to obtain high-quality starting solutions and a learning-based strategy to select suitable perturbations. Experimental results on 30 benchmark instances show the competitiveness of the algorithm against the state-of-the-art methods, by reporting 18 new lower bounds and 12 equal results (including 7 known optimal results). We also perform additional experiments to validate the design of the algorithmic components.

Keywords: Learning-driven search; Heuristic; Partial set covering; Fast neighborhood evaluation.

* Corresponding author.

Email addresses: `101012533@seu.edu.cn` (Wen Sun), `220215468@seu.edu.cn` (Wenlong Li), `hao@info.univ-angers.fr` (Jin-Kao Hao), `qinghuawu1005@gmail.com` (Qinghua Wu).

1 Introduction

The profit maximization set covering problem (PMSCP) was introduced in [3] to formulate the drill-hole placement problem in mining industry. In this application, the candidate drill holes are grouped by different drill stations where the drill-platform can be installed and each candidate drill hole covers a number of blocks. There is a cost of drilling a hole that proportional to the hole length and a gain generated by each block covered by a hole. Moreover, displacing the drill-platform from one station to another implies a high cost. We want to perform a number of drill holes that can maximize the difference of the total gain of the blocks covered by the performed drill holes subtracting the costs of drilling the holes and moving the drill-platform. Then, the problem can be formally defined by the following model and the details of practical applications can be found in [3].

Let $E = \{e_1, e_2, \ldots, e_m\}$ denote a set of m elements, and $S = \{s_1, s_2, \ldots, s_n\}$ a collection of n non-empty subsets of E, which are grouped into q groups and let $G = \{g_1, g_2, \ldots, g_q\}$ be the union of these q groups. Moreover, each element e_i $(i = 1, \ldots, m)$ of E has a gain or profit a_i; each subset s_j $(j = 1, \ldots, n)$ of S has a cost b_j; each group g_k $(k = 1, \ldots, q)$ of G has a cost c_k. The PMSCP is to select a collection X of some subsets from S to maximize the objective function, which is the difference of the total gain of the covered elements subtracting the total cost of the selected subsets and the groups containing these subsets. The gain of a covered element is counted once, even though the element is part of several selected subsets; similarly, the cost of a group is counted only once, even though the group contains several selected subsets.

Formally, let U be a binary matrix (called element-set matrix) specifying the relationship between the elements of E and the subsets of S such that $u_{ij} = 1$ if element e_i appears in subset s_j and else $u_{ij} = 0$. Let V be a binary matrix (called group-set matrix) specifying the relationship between the subsets of S and the groups of G such that $v_{jk} = 1$ if subset s_j belongs to group g_k and else $v_{jk} = 0$. Let x_j be a binary variable (decision variable) such that $x_j = 1$ if subset s_j is selected and else $x_j = 0$. Let y_i be a binary variable (logical variable) such that $y_i = 1$ if e_i is covered by a selected subset and else $y_i = 0$. Let z_k be a binary variable (logical variable) such that $z_k = 1$ if g_k contains at least one selected subset and else $z_k = 0$. Let $X = (x_1, x_2, \ldots, x_n)$ be a binary vector representing a candidate solution. Then the PMSCP can be stated as follows [3].

$$(PMSCP) \qquad \text{Maximize} \quad f(X) = \sum_{i=1}^{m} a_i y_i - \sum_{j=1}^{n} b_j x_j - \sum_{k=1}^{q} c_k z_k \qquad (1)$$

$$\text{subject to} \quad y_i \leq \sum_{j=1}^{n} u_{ij} x_j, \; i \in \{1, ..., m\} \tag{2}$$

$$v_{kj} x_j \leq z_k, \; j \in \{1, ..., n\}, \; k \in \{1, ..., q\} \tag{3}$$

$$x_j, y_i, z_k \in \{0, 1\} \tag{4}$$

Equation (1) (objective function) is to maximize the difference of the total profit of the elements covered by the selected subsets in X subtracting the costs of the chosen subsets and the costs of the groups containing the selected subsets. Constraint (2) ensures that an element is covered if it appears in one or more selected subsets. Constraint (3) states that a group containing at least one selected subset is part of the solution. Constraint (4) stipulates the binary nature for variables x_j, y_i and z_k.

Figure 1 provides an illustrative example of the PMSCP with $E = \{e_1, e_2, ..., e_6\}$, $S = \{s_1, s_2, ..., s_5\}$, and $G = \{g_1, g_2\}$. The element gains and subset/group costs are indicated next to each item as well as the element-set matrix U and group-set matrix V. Figure 1(c) shows a candidate solution $X = \{s_4\}$ that covers three elements $\{e_1, e_4, e_5\}$ and belongs to group g_2. This solution has an objective value of 6, since the total gain of the covered elements is (5+3+6) (orange elements), the cost of the selected subset is 5 (orange subset) and the cost of the corresponding group is 3 (orange group). Figure 1(d) exemplifies an optimal solution $X = \{s_1, s_2\}$ with the maximum objective value of 9.

Now it is easy to observe that the above drill-hole placement problem is an instance of the PMSCP, if we consider that E is the set of blocks with their gains, S is the set of candidate drill holes with their costs, and G is composed of the drill-platform locations with their moving costs.

The PMSCP is tightly related to the popular set covering problem (SCP) [12] and can be considered as a profit-maximization variant of the partial set covering problem where it is unnecessary to cover all elements of E.

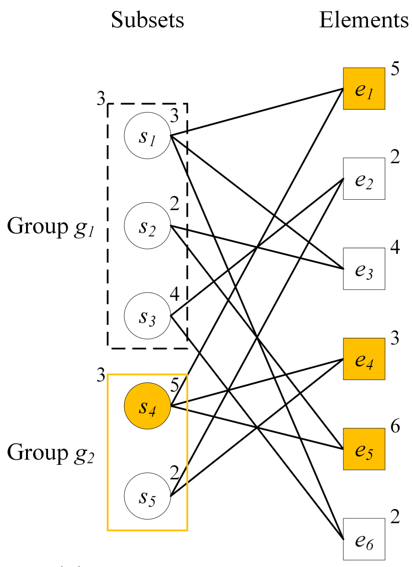
Like the related SCP and its various variants, the PMSCP is computationally challenging. Bilal et al. [3] introduced an iterated tabu search (ITS) that integrates a tabu search and an adaptive perturbation mechanism, which was tested on 30 benchmark instances derived from the drill-hole placement problem. Along with the ITS algorithm, the exact branch and bound algorithm (B&B) of the CPLEX solver (CPLEX (B&B)) for solving mixed integer programming problems was run for a long computation time (up to 9 days per instance) to solve the mathematical model (1)-(4). Results show that CPLEX (B&B) can obtain 7 optimal solutions and 13 feasible solutions, but fails to find a feasible solution for the other instances. Compared with CPLEX (B&B), ITS obtains 22 better, 5 equal, and 3 worse results. Bilal et al. [3] also compared ITS with an adaptation to the PMSCP of the memetic algorithm (MA-PMSCP) initially designed for the SCP in [2], demonstrating the dominance

$$U = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \end{bmatrix}$$

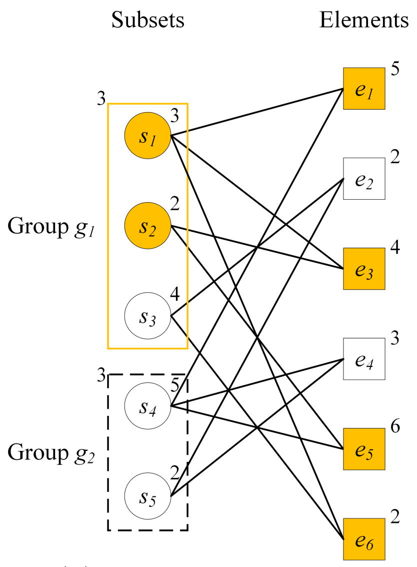
(a) An element-set matrix U

$$V = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix}$$

(b) A group-set matrix V



(c) A candidate solution



(d) An optimal solution

Fig. 1. An example of PMSCP with its element-set matrix U, group-set matrix V, a candidate solution and an optimal solution

of ITS over MA-PMSCP. In [4], a Parallel ITS algorithm was proposed, which combines the ITS and a parallel neighborhood evaluation approach. Experimental results showed that Parallel ITS is superior to ITS by obtaining 20 better, 8 equal and 2 worse results. We notice that CPLEX (B&B), ITS and Parallel ITS are the best existing approaches for solving the PMSCP and will be used as the references for our experimental study.

According to the reported results, existing heuristic algorithms for the PMSCP are unstable, especially on large instances with at least 5192 elements, 15252 subsets, and 62 groups. Besides, these algorithms are primarily based on conventional ITS approach and ignore powerful learning-based frameworks. Indeed, it is recently shown that algorithms integrating learning techniques and conventional heuristic approaches can achieve top performance for covering problems such as the set covering problem [6] and the budgeted maximum coverage problem [15]. These observations motivate us to devise a learning-based heuristic algorithm that is able to solve large-scale instances.

In this paper, we introduce the first learning-based local search algorithm for the PMSCP with the following contributions.

First, we design a two-phase iterated local search that employs an intensification-driven search strategy to perform an intensified examination of candidate solutions, fast neighborhood evaluations to shorten the computation time, and a two-phase local search to effectively examine the search space.

Second, we present a learning-driven construction method to generate high-quality initial solutions by using historical information of the discovered local optimal solutions, and an informed perturbation to dynamically select a proper perturbation.

Finally, we show 18 record-breaking best solutions among the 30 benchmark instances, which can be used for the problem in future research.

Section 2 introduces the proposed algorithm. Section 3 shows comparative results. Section 4 justifies the design of the key algorithmic components, and the last section draws conclusions and presents research perspectives.

2 Learning-based multi-start iterated local search

The proposed learning-based multi-start iterated local search algorithm (LMILS) for the PMSCP relies on the multi-start iterated local search framework [18] that iterates solution construction and local optimization. As a general approach, multi-start iterated local search has been applied to several hard optimization problems including the set-union knapsack [26], maximum clique [27], bandwidth coloring [14] and minimum weight vertex cover [28]. In this work, we propose the first adaptation of the multi-start iterated local search to the PMSCP enhanced by learning techniques. Algorithm 1 presents the pseudo-code of LMILS.

The LMILS algorithm (Algorithm 1) begins with a reduction procedure to pre-process the given input instance (line 1). Then a number of iterations are performed until the stopping condition is satisfied. At each iteration, an initial solution X_0 is generated by the learning-driven initialization procedure (line 6) and then improved by the intensification-driven iterated local search procedure (line 7). The recorded best solution (X^*) is updated whenever necessary, and finally returned as the output at the end of the LMILS algorithm. In the following section, we illustrate the details of the reduction-based pre-processing procedure, the learning-driven construction procedure, and the two-phase local optimization procedure.

Algorithm 1 Framework of the LMILS algorithm

```
Input: Instance $I = (E, S, G)$, greediness ratio $\epsilon$, depth of iterated local search $\omega$,
    depth of tabu search $\beta$
Output: The best solution $X^*$ ever found
 1: $I_0 \leftarrow Reduction(I)$                                    /*See Section 2.1*/
 2: $X^* \leftarrow \emptyset$                              /*Initialize the best solution $X^*$*/
 3: Initialize the historical information vector $\eta$              /*See Section 2.2*/
 4: Initialize the perturbation probability vector $\gamma$          /*See Section 2.3.2*/
 5: while Stopping condition is not satisfied do
 6:     $X_0 \leftarrow Learning\_Driven\_Initialization(I_0, \eta, \epsilon)$
                              /*Generate an initial solution $X_0$, see section 2.2*/
 7:     $(X_0, \eta, \gamma) \leftarrow Intensification\_Driven\_Iterated\_Local\_Search(X_0, \omega, \beta, \eta, \gamma)$
                              /*Improve the initial solution $X_0$, see section 2.3*/
 8:     if $f(X_0) > f(X^*)$ then
 9:         $X^* \leftarrow X_0$
10:     end if
11: end while
12: return $X^*$
```

2.1 Reduction-based pre-processing

We first use a reduction procedure to pre-process the given problem instance $I = (E, S, G)$ before the search starts, in which a subset of S can be eliminated if the total gain of the elements covered by this subset is no more than the cost of the subset. This is achieved by traversing all subsets of S and removing any subset s_j if $\sum_{e_i \in s_j} a_i \leq b_j$.

Figure 2 presents an example of the reduction procedure. The cost of subset s_3 is 4, which is equal to the total gain of its covered elements (e_2 and e_6). Thus, s_3 is removed from S. In Appendix A, we investigate the impact of the reduction procedure (in fact, it can reduce the size of S by 5%-30%).

Let S_0 be the reduced S after this reduction procedure and let $I_0 = (E, S_0, G)$ denote the pre-processed instance. The LMILS algorithm starts its search by obtaining an initial solution with the learning-driven solution construction procedure.

2.2 Learning-driven solution construction

Multi-start iterated local search typically generates a new starting solution from the previous (best) local optimal solution by performing some perturbation operations. In this work, we devise a learning-driven solution construction procedure to generate promising new starting solutions with the help of historical information. This is in sharp contrast to the non-informed perturbations

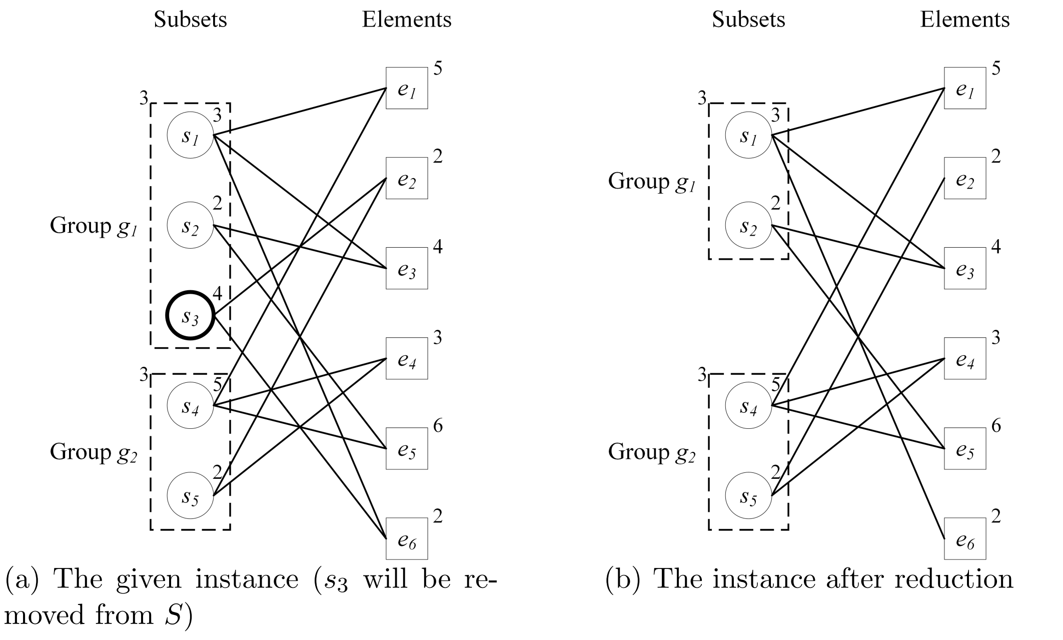


Fig. 2. An example of the reduction procedure

adopted in [3,4].

Algorithm 2 Learning-driven initialization

```
Input: Reduced instance $I_0 = (E, S_0, G)$, historical information vector $\eta$, greediness ratio $\epsilon$
Output: The initial solution $X_0$
 1: $X_0 \leftarrow \emptyset$
 2: $S' \leftarrow S_0$     /*Initialize the set $S'$ of the subsets that have not been chosen */
 3: while $S' \neq \emptyset$ do
 4:     /*Choose a subset $s_j$*/
 5:     if $rand(0,1) < \epsilon$ then
 6:         Choose a subset $s_j \in S'$ with the largest value $\eta_j$ in vector $\eta$
 7:     else
 8:         Randomly choose a subset $s_j \in S'$
 9:     end if
10:     $S' \leftarrow S' \backslash \{s_j\}$                                        /*Update $S'$*/
11:     /*Insert the chosen subset $s_j$*/
12:     if $\sum_{e_i \in s_j \wedge e_i \notin \cup_{s_l \in X_0} s_l} a_i - b_j > 0$ and $rand(0,1) < \eta_j$ then
13:         $X_0 \leftarrow X_0 \cup \{s_j\}$
14:     end if
15: end while
16: return $X_0$
```

The learning-driven solution construction iteratively inserts one subset from S_0 to the current solution X_0 according to the historical information vector $\eta = \{\eta_1, \eta_2, ..., \eta_n\}$ (lines 3-15, Algorithm 2), where η_j records the probability of s_j being inserted into the solution X_0. η_j is initialized to 0.5, $\forall j \in \{1, 2, ..., n\}$

(line 3, Algorithm 1), X_0 to $\emptyset$ (line 1, Algorithm 2), and S' to S_0 that represents the set of unchosen subsets (line 2, Algorithm 2). At each iteration, first, the subset $s_j \in S'$ with the largest value η_j in vector η is chosen with a probability of ϵ, otherwise, a random subset $s_j \in S'$ is chosen (lines 5-9, Algorithm 2); then, the chosen s_j is inserted into X_0 if its incremental gain is higher than its cost and η_j is more than a random probability (line 12, Algorithm 2); otherwise, s_j is ignored. The construction procedure stops when S' becomes empty.

2.3 Intensification-driven iterated local search

Starting from the solution brought by the learning-driven construction procedure (Section 2.2), the MLILS algorithm uses the intensification-driven iterated local search (IILS) (Algorithm 3) to find improved solutions. Specifically, IILS iterates the two-phase local search procedure line 5) and the learning-based perturbation procedure (line 14) until it cannot improve its best solution for ω consecutive iterations (ω is the depth of iterated local search).

Algorithm 3 Intensification-driven iterated local search

```
Input: Initial solution $X_0$, the depth of IILS $\omega$, the depth of tabu search $\beta$, probability vectors $\eta$ and $\gamma$
Output: The best solution $X_{best}$ found during this search process, updated probability vectors $\eta$ and $\gamma$
 1: $X \leftarrow X_0$                                  /*Initialize the current solution $X$*/
 2: $X_{best} \leftarrow X$                    /*Initialize the best solution $X_{best}$ of the IILS*/
 3: $non\_improve \leftarrow 0$     /*Indicate the consecutive iterations where $X_{best}$ is not updated*/
 4: while $non\_improve < \omega$ do
 5:    $X \leftarrow TwoPhase\_LocalSearch(X, \beta)$                /*See Section 2.3.1*/
 6:    if $f(X) > f(X_{best})$ then
 7:       Update historical information vector $\eta$ according to Equation (13) and (14)
 8:       Update perturbation probability vector $\gamma$ according to Equation (15)
 9:       $X_{best} \leftarrow X$                  /*Update the best solution $X_{best}$ of IILS*/
10:       $non\_improve \leftarrow 0$
11:    else
12:       $non\_improve \leftarrow non\_improve + 1$
13:    end if
14:    $X \leftarrow Perturbation(X_{best}, \gamma)$                 /*See Section 2.3.2*/
15: end while
16: return $(X_{best}, \eta, \gamma)$
```

2.3.1 Two-phase local search

The two-phase local optimization aims to improve the input solution as described in Algorithm 4. The first phase, called the flip-based tabu search, performs an exploratory and fast search (line 5), while the second phase, called the swap-based descent search, performs a more focused search (line 6). The two phases alternate until no further improvement is possible.

Algorithm 4 Two-phase local search

```
Input: Input solution $X$, the depth of tabu search $\beta$
Output: Updated solution $X$
 1: $X_b \leftarrow X$        /*Initialize the best solution $X_b$ of the two-phase local search*/
 2: $non\_improve \leftarrow 0$        /*Indicate whether $X$ has been improved */
 3: while $non\_improve = 0$ do
 4:     $non\_improve \leftarrow 1$
 5:     $X \leftarrow Flip\_TabuSearch(X, \beta)$        /*See Section 2.3.1.1*/
 6:     $X \leftarrow Swap\_DescentSearch(X)$        /*See Section 2.3.1.2*/
 7:     if $f(X) > f(X_b)$ then
 8:         $X \leftarrow X_b$   /*Update the best solution $X_b$ of the two-phase local search*/
 9:         $non\_improve \leftarrow 0$
10:     end if
11: end while
12: return $X$
```

2.3.1.1 Flip-based tabu search Starting with an input solution X, the flip-based tabu search procedure iteratively improves the solution according to the tabu search method [9], which explores the flip-based neighborhood (see below). At each iteration, the best admissible candidate solution among the neighboring solutions is taken to replace the current solution. The corresponding flip move is recorded in the so-called tabu list to forbid the reverse move for a number of subsequent iterations. The tabu search process continues until its best solution cannot be further improved for a number of consecutive iterations.

Flip Move Operator. Following [3], we apply the flip operator to generate neighboring solutions. Generally, this operator changes the value of a binary variable to its complementary value. In our case, $flip(X, j)$ adds a subset $s_j \notin X$ to the current solution X or removes a subset s_j from X. Let $X \oplus flip(X, j)$ designate the neighboring solution given by applying $flip(X, j)$ to X. The neighborhood N_F induced by the $flip(X, j)$ operator is given by

$$N_F(X) = \{X \oplus flip(X, j) : s_j \in S_0\}. \tag{5}$$

It can be seen easily that the size of N_F is limited by $O(|S_0|)$.

Fast Neighborhood Evaluation Technique. We design a fast incremental evaluation technique to efficiently compute the move gain (Δf), which identifies the variation of the objective value f (Equation (1)) after applying the $flip(X, j)$ operator. We maintain a vector δ of size n, $\delta[j]$ $(1 \leq j \leq n)$, for each $s_j \in g_k$ to record the incremental evaluation value of flipping s_j. We can initialize the vector δ as follows

$$\delta[j] = \begin{cases} \sum_{e_i \in s_j, |R_i \cap X| = 0} a_i - b_j - \theta_1 * c_k, & s_j \notin X \\ -\sum_{e_i \in s_j, |R_i \cap X| = 1} a_i + b_j + \theta_2 * c_k, & s_j \in X \end{cases} (1 \leq i \leq m, 1 \leq k \leq q) \tag{6}$$

where $\theta_1 = 1$ if no subset is shared by g_k and X, i.e., $g_k \cap X = \emptyset$; otherwise, $\theta_1 = 0$; $\theta_2 = 1$ if only one subset is shared by g_k and X, i.e., $g_k \cap X = \{s_j\}$; otherwise, $\theta_2 = 0$; e_i denotes an element of the subset s_j, and R_i represents the union of such subsets.

Each time a $flip(X, j)$ move involving subset s_j is performed, the move gain $\Delta f(flip(X, j))$ is obtained by,

$$\Delta f(flip(X, j)) = f(X \oplus flip(X, j)) - f(X) = \delta[j]. \tag{7}$$

When the flip move is performed, the affected data structure δ is updated as follows.

1) For the flipped subset s_j, $\delta[j] = -\delta[j]$.

2) For each subset s_l in the same group g_k as the flipped s_j $(l \neq j)$, $\delta[l]$ is updated by

$$\delta[l] = \begin{cases} \delta[l] - c_k, \; if \; (X \cap g_k = \{s_j\} \; and \; s_j \in X) \; or \; (X \cap g_k = \{s_l\} \; and \; s_j \notin X); \\ \delta[l] + c_k, \; if \; (X \cap g_k = \{s_l, s_j\} \; and \; s_j \in X) \; or \; (X \cap g_k = \emptyset \; and \; s_j \notin X); \end{cases} \tag{8}$$

3) For each subset s_r covering the element e_i in s_j $(r \neq j)$, the value $\delta[r]$ of s_r

is updated by,

$$\delta[r] = \begin{cases} \delta[r] + a_i, \ if \ (X \cap R_i = \{s_j\} \ and \ s_j \in X) \ or \ (X \cap R_i = \{s_r\} \ and \ s_j \notin X); \\ \delta[r] - a_i, \ if \ (X \cap R_i = \{s_r, s_j\} \ and \ s_j \in X) \ or \ (X \cap R_i = \emptyset \ and \ s_j \notin X); \end{cases} \tag{9}$$

Clearly, the time complexity of updating δ is $O(mn)$ in the worst case.

Tabu List Management Strategy. To prevent the search from revisiting recently encountered solutions, we adopt an n-vector T as our tabu list to record the subsets involved in the flip operator, where $T[j]$ is initially set to 0 and is updated as follows each time a subset s_j is flipped in the current solution X,

$$T[j] = \begin{cases} Iter + rand(C), \ if \ s_j \ is \ added \ to \ X; \\ Iter + |S_0|, \qquad if \ s_j \ is \ removed \ from \ X; \end{cases} \tag{10}$$

where $Iter$ is the current number of iterations, $rand(C)$ is a random integer between 1 to C. C is the tabu tenure threshold set to 5 according to experiment (see Section 3.2), and $|S_0|$ is the number of the available subsets after the reduction procedure. Then, during the next iterations, if $Iter < T[j]$, s_j is forbidden to take part in any flip operation, unless the flip involving s_j leads to a neighboring solution that is better than the recorded best solution X_b of the tabu process (this exception rule is called an aspiration criterion in tabu search [9]).

The framework of the proposed tabu search is illustrated in Algorithm 5. Starting with the input solution X (line 1), it enters a while loop to iteratively improve the current solution (lines 3-13). At each iteration, the algorithm selects one best admissible neighboring solution X' in the neighborhood $N_F(X)$ to take it as the new current solution (lines 4 and 5), records the underlying flip move in the tabu list (line 6), conditionally renews the best recorded solution X_b, and updates the consecutive non-improvement counter (lines 7-11). Tabu search terminates when the best solution found during this search is not updated for β continuous iterations (β is the depth of tabu search).

2.3.1.2 Swap-based descent search The swap-based descent search (Algorithm 6) aims to find still better solutions by using an enlarged neighborhood N_S induced by the swap operator. At each iteration of the descent search, the current solution X is substituted by the best neighboring solution X' found in $N_S(X)$, if X' is better than X (lines 2-9). Swap-based descent search stops when a local optimal solution is reached in the swap neighborhood.

Algorithm 5 Flip-based tabu search

```
Input: Input solution $X$, depth of tabu search $\beta$
Output: The best solution $X_b$ found during the tabu process
 1: $X_b \leftarrow X$        /*Initialize the best solution $X_b$ of the flip-based tabu search*/
 2: $non\_improve \leftarrow 0$        /*Indicate the continuous iterations where $X_b$ is not updated*/
 3: while $non\_improve \leq \beta$ do
 4:     Choose a best admissible neighboring solution $X' \in N_F(X)$
 5:     $X \leftarrow X'$
 6:     Update the tabu list
 7:     if $f(X) > f(X_b)$ then
 8:         $X_b \leftarrow X$     /*Update the best solution $X_b$ of the flip-based tabu search*/
 9:         $non\_improve \leftarrow 0$
10:     else
11:         $non\_improve \leftarrow non\_improve + 1$
12:     end if
13: end while
14: return $X_b$
```

Algorithm 6 Swap-based descent search

```
Input: Input solution $X$
Output: The improved solution $X$
 1: $non\_improve \leftarrow 0$        /*Indicate whether $X$ has been improved*/
 2: while $non\_improve = 0$ do
 3:     $non\_improve \leftarrow 1$
 4:     Choose the best neighboring solution $X' \in N_S(X)$
 5:     if $f(X') > f(X)$ then
 6:         $X \leftarrow X'$        /*Update the current solution $X$*/
 7:         $non\_improve \leftarrow 0$
 8:     end if
 9: end while
10: return $X$
```

Swap Move Operator. The swap operator exchanges a selected subset s_j from X with an unselected subset s_l, leading to the following N_S neighborhood

$$N_S(X) = \{X \oplus swap(X, j, l) : s_j \in X, s_l \notin X\}. \tag{11}$$

The size of N_S is bounded by $O(|X| \times |S_0 \setminus X|)$.

Fast Neighborhood Evaluation Technique. The $swap(X, j, l)$ move can be performed as a combination of two consecutive $flip$ moves, i.e., $X \oplus swap(X, j, l) = (X \oplus flip(X, j)) \oplus flip(X \oplus flip(X, j), l)$. As a result, we can compute the move value $\Delta f(swap(X, j, l))$ as $\Delta f(swap(X, j, l)) = \delta[j] + \delta'[l]$, where $\delta[j]$ is the incremental objective value of flipping s_j and $\delta'[l]$ is the incremental objective value of flipping s_l after flipping s_j.

We initialize $\delta[j]$ by Equation (6), and $\delta'[l]$ by

$$\delta'[l] = \begin{cases} \delta[l] - c_k, \; if \; X \cap g_k = \{s_j\} \; and \; s_l \in g_k \cap (S_0 \setminus X); \\ \delta[l] + a_i, \; if \; X \cap R_i = \{s_j\} \; and \; s_l \in R_i \cap (S_0 \setminus X); \\ \delta[l], \qquad otherwise. \end{cases} \tag{12}$$

When a swap move is performed, the affected data structure δ can be updated according to the updating rule of the $flip$ move (see Section 2.3.1.1). δ' will be recalculated accordingly. The complexity of the swap move is $O(mn)$.

2.3.2 Learning-based perturbation strategy

The perturbation procedure (Algorithm 7) aims to jump from the current search zone to other areas. Our perturbation procedure employs two types of perturbations: *Set_Perturbation* and *Group_Perturbation*, which shares the ideas of the ruin-and-recreate strategy [22]. For each perturbation iteration, one type is selected with a learning-based strategy inspired by the technique from [17]. The perturbation probability vector γ records the possibility γ_t of the t-th type of perturbation being selected, and $\sum_{t=1}^{2} \gamma_t = 1$. At first, the probability of each perturbation type being selected is the same, i.e, $\gamma_t = 1/2$. Then, a random number "*rand*" between 0 and 1 is generated. If the $rand < \gamma_1$, the *Set_Perturbation* (the 1st type of perturbation) is selected (lines 1-2); otherwise, the *Group_Perturbation* (the 2nd type of perturbation) is selected (lines 3-4).

Algorithm 7 Perturbation procedure

```
Input: The best solution in IILS $X_{best}$, the perturbation probability vector $\gamma = \{\gamma_1, \gamma_2\}$
Output: The perturbed solution $X$
1: if $rand(0,1) < \gamma_1$ then
2:     $X \leftarrow Set\_Perturbation(X_{best})$
3: else
4:     $X \leftarrow Group\_Perturbation(X_{best})$
5: end if
6: return $X$
```

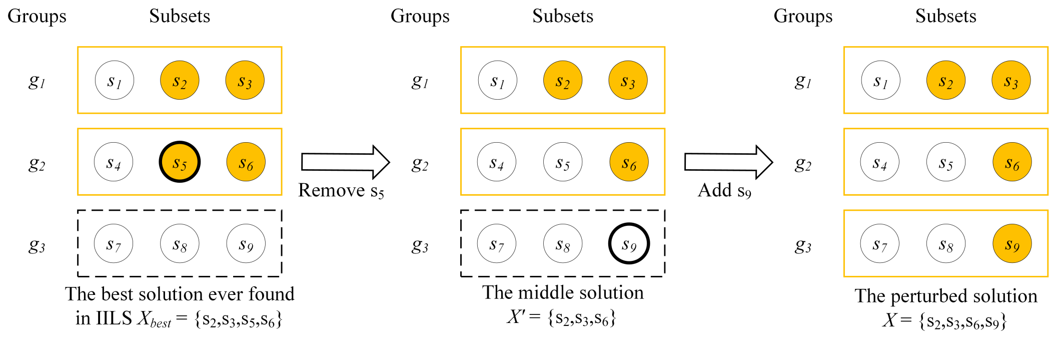
We now explain the two perturbations as follows.

1) *Set_Perturbation*: This perturbation first removes all subsets of the best solution X_{best} of IILS with a probability p ($p = 0.3$ in this work). Let h be the number of removed subsets. Then, randomly choose h subsets from S_0 and add each chosen subset if it is not in the solution X_{best}; otherwise, skip it.

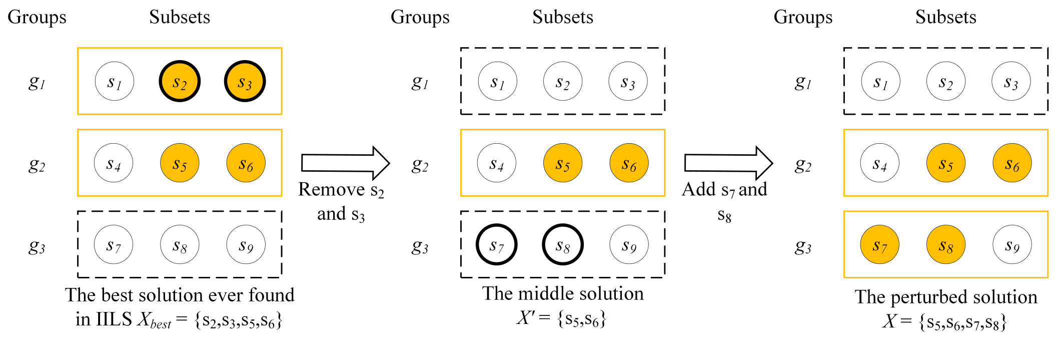
2) *Group_Perturbation*: This perturbation first identifies a collection of groups

$Z = \{g_k | g_k \cap X \neq \emptyset, g_k \in G\}$ where the intersection of each group with X_{best} is not empty. Then randomly choose $max\{p \times |Z|, 1\}$ groups from Z, and remove the intersection subsets between the chosen groups and X_{best}. Finally, randomly choose $max\{p \times |Z|, 1\}$ groups from $G \backslash Z$, and randomly flip $\lfloor |X_{best}|/|Z| \rfloor$ subsets of each chosen groups.

Figure 3 illustrates the two perturbations. Figure 3(a) shows the set perturbation on the solution $X_{best} = \{s_2, s_3, s_5, s_6\}$. It first removes all subsets of X_{best} with the probability $p = 0.3$. Suppose that only one subset s_5 from X_{best} is removed, then we randomly choose one subset from S_0, say s_9. Because the chosen subset s_9 is not in X_{best}, it is added, leading to the perturbed solution $X = \{s_2, s_3, s_6, s_9\}$. Figure 3(b) illustrates the group perturbation on $X_{best} = \{s_2, s_3, s_5, s_6\}$. This perturbation randomly chooses a group g_1 from $Z = \{g_1, g_2\}$, and removes the intersection subsets s_2 and s_3 between g_1 and X_{best}. Then, it randomly chooses a group g_3 from $G \backslash Z = \{g_3\}$. Finally, by flipping two random subsets ($|X_{best}|/|Z| = 2$) (say, s_7 and s_8) in the chosen group g_3, a perturbed solution $X = \{s_5, s_6, s_7, s_8\}$ is obtained.


Groups
Subsets
g_1
g_2
g_3
s_1 s_2 s_3
s_4 s_5 s_6
s_7 s_8 s_9
Remove s_5
Add s_9
The best solution ever found in IILS $X_{best} = \{s_2,s_3,s_5,s_6\}$
The middle solution $X' = \{s_2,s_3,s_6\}$
The perturbed solution $X = \{s_2,s_3,s_6,s_9\}$

(a) The example of *Set_Perturbation*


Groups
Subsets
g_1
g_2
g_3
s_1 s_2 s_3
s_4 s_5 s_6
s_7 s_8 s_9
Remove s_2 and s_3
Add s_7 and s_8
The best solution ever found in IILS $X_{best} = \{s_2,s_3,s_5,s_6\}$
The middle solution $X' = \{s_5,s_6\}$
The perturbed solution $X = \{s_5,s_6,s_7,s_8\}$

(b) The example of *Group_Perturbation*

Fig. 3. Examples of *Set_Perturbation* and *Group_Perturbation*

The perturbed solution is given to the intensification-driven iterated local search (Algorithm 3) to start the next round of search.

2.4 Historical information update

As mentioned above, LMILS employs the historical information vector η to construct initial solutions (Section 2.2) and perturbation probability vector γ to select the type of perturbation (Section 2.3.2). Both vectors are updated each time the best solution X_{best} in the IILS procedure is improved.

Update the historical information vector η. The inspiration for updating the vector η comes from the learning automata [20], ant system [7], and max-min ant system [23]. The principle is to raise the probability of subsets being selected by while reducing that of taken out from the current solution X if X is better than X_{best}. Specifically, we modify the probability value of each subset as follows,

$$\eta_j = \begin{cases} \phi_1 + (1-\phi_1) \times \eta_j, \; if \; s_j \in X \; and \; s_j \in X_{best} \\ \phi_2 + (1-\phi_2) \times \eta_j, \; if \; s_j \in X \; and \; s_j \notin X_{best} \\ (1-\phi_1) \times \eta_j, \qquad if \; s_j \notin X \; and \; s_j \notin X_{best} \\ (1-\phi_2) \times \eta_j, \qquad if \; s_j \notin X \; and \; s_j \in X_{best} \end{cases} (1 \leq j \leq n) \tag{13}$$

where reward factors ϕ_1 $(0 < \phi_1 < 1)$ and ϕ_2 $(0 < \phi_2 < 1)$ are respectively set to 0.2 and 0.3 (see Section 3.2).

Besides, a probability smoothing technique is adopted to prevent historical information from misleading the current search. Once a subset in the historical information vector achieves the given probability threshold, it will be increased or decreased by a smoothing factor (i.e., $\phi_3, 0 < \phi_3 < 1$) to ignore some earlier information. The details are as follows.

$$\eta_j = \begin{cases} \phi_3 + (1-\phi_3) \times \eta_j, \; if \; \eta_j < 1-\alpha \\ (1-\phi_3) \times \eta_j, \qquad if \; \eta_j > \alpha \end{cases} (1 \leq j \leq n) \tag{14}$$

where smoothing threshold α is a parameter.

Update the perturbation probability vector γ. The perturbation probability vector $\gamma = \{\gamma_1, \gamma_2\}$ is updated following a probability learning method (Equation (15)). The inspiration for updating the probability in this work is from [16,24], which is helpful to determine the optimal candidate strategies.

$$\gamma_t = \frac{d_0 + d_t}{2 \times d_0 + d_1 + d_2} \tag{15}$$

t is the perturbation type, d_0 is an adjustment parameter and d_t is the frequency that local optimal solution X_{best} is updated by adopting the t-type perturbation. If d_t increases in the current iteration, the numerator and the denominator of the Equation (15) will both increase with the same quantity. The γ_t value rises, while the selection probability of the other perturbation accordingly decreases.

2.5 Computational complexity

The time complexity of the learning-driven solution construction procedure is bounded by $O(n^2)$. As to the intensification-driven iterated local search, its two-phase local search requires $O(K_3 \times (K_1 + K_2 \times n) \times mn)$ time, where K_1 is the number of iterations of the tabu search, K_2 is the number of iterations of the descent search, and K_3 is the number of iterations of the two-phase local search, whereas its perturbation phase requires $O(n)$ time. Therefore, one iteration of the main loop of the LMILS algorithm has a complexity of $O(K_4 \times K_3 \times (K_1 + K_2 \times n) \times mn)$, where K_4 is the number of iterations of the intensification-driven iterated local search.

3 Computational assessment and comparative analysis

We test the LMILS algorithm on the 30 realistic benchmark instances from the mining industry that were initially proposed by Bilal et al. [3]. Among them, the optimal objective value is known for 7 instances and given in [3]. For the other instances, the best results (best lower bounds) were obtained in [3] and [4].

3.1 Benchmark instances

We classify the 30 benchmark instances [1] into three categories. The term BKV (the best-known value) stands for the optimal objective value (if known) or the best lower bound in the literature.

[1] Available at https://github.com/sunseu2022/PMSCP/.

1. The first category consists 10 small instances with A* and B* in their names, among which the optimal values of 2 instances (A1 and A2) are known. These instances feature the number of elements $m = 1000$, the number of subsets n within $[3493, 82635]$, the number of groups $q \in \{10, 20\}$, the element gain within $[100, 300]$, the subset cost within $[300, 1500]$, and the group cost 1000.
2. The second category contains 10 medium instances whose names start with C* or D*, where the optimal values of 5 instances (C1,C2,C3,C4 and C5) are known. Each instance has $m = 5192$ elements, $n \in \{6633{,}117274\}$ and $q \in \{31, 62\}$ groups. Besides, each element has a gain within $[1, 100]$, each subset has a cost within $[100, 1200]$, and each group has a cost of 1000.
3. The third category contains 10 large instances with E* or F* in their names. Each instance is characterized by its number of elements $m = 15625$, number of subsets n within $[10325, 462666]$, and number of groups $q \in \{96, 100\}$. Each element has a gain within $[1, 100]$ and each subset has a cost within $[500, 1500]$. The cost of each group is equal to 500 or 1000.

3.2 Experimental Protocol

The proposed algorithm was coded in C++ [2] and compiled by GNU g++ 4.1.2 with the -O3 flag. All experiments were conducted on an Intel (R) Core (TM) 2 Duo CPU T7700 2.40GHz processor with 2GB of RAM running on the Ubuntu CentOS Linux release 7.9.2009 (Core).

Parameters: LMILS requires 10 parameters: greediness ratio ϵ in initialization phase; search depth of IILS ω, search depth of flip-based tabu search β, tabu tenure threshold C of flip-based tabu search, perturbation strength p in the IILS phase; reward factors ϕ_1 and ϕ_2, smoothing coefficient ϕ_3, smoothing threshold α for the probability vector η; and adjustment parameter d_0 of the probability vector γ in the historical information update phase. We used the "irace" package to automatically determine the fittest parameter settings. The "irace" was run on 18 representative instances (relatively difficult as shown in Tables 4-6) with a budget of 200 executions. Table 1 presents both the candidate parameter values and final parameter values suggested by "irace". All experiments employed the final parameter values.

Reference algorithms: For comparison, we used 3 state-of-the-art algorithms.

(1) The CPLEX solver (B&B) [3] (run on an Intel(R)Xeon(R) X7550 with a 2.00GHz processor and 1 TB of RAM);

[2] The binary code is available at https://github.com/sunseu2022/PMSCP/.

Table 1
Parameter settings of the LMILS algorithm

Parameters	Section	Description	Candidate values	Final value
ϵ	2.2	The greediness ratio of initialization	[0.1,0.9]	0.6
ω	2.3	The search depth of IILS	{100, 200, 300, 400, 500}	300
β	2.3.1.1	The search depth of flip-based tabu search	{500, 1000, 1500, 2000, 2500}	1000
C	2.3.1.1	The tabu tenure threshold of flip-based tabu search	{1, 5, 10, 15, 20}	5
p	2.3.2	The perturbation strength	{0.1, 0.2, 0.3, 0.4, 0.5}	0.3
ϕ_1	2.4	The first reward factor of probability vector η	[0.1,0.9]	0.2
ϕ_2	2.4	The second reward factor of probability vector η	[0.1,0.9]	0.3
ϕ_3	2.4	The smoothing coefficient of probability vector η	[0.1,0.9]	0.3
α	2.4	The smoothing threshold of probability vector η	{0.8, 0.85, 0.9, 0.95, 0.99}	0.95
d_0	2.4	The adjustment parameter of probability vector γ	{1, 25, 50, 75, 100}	50

(2) Iterated tabu search algorithm (ITS) [3] (run on an Intel(R)Xeon(R) X7550 with a 2.00GHz processor and 1 TB of RAM);
(3) Parallel ITS [4] (run on two Intel Xeon E5-2697 v2 with 2.7GHz processors and 64 GB of RAM.

Stopping conditions: For our comparative study, we focused on the quality of the solutions found. Nevertheless, we also provide the details of computation time for indication purpose. Similar to [25,5], the CPU frequency of our computer (Intel Core T7700 2.40GHz) was taken as the basis to scale the time of the reference algorithms (see Table 2 for the scaling factors). We made sure that our stopping conditions are comparable to those used by the reference algorithms.

Table 2
Scaling factors for the computers used by reference algorithms. Our computer (Intel Core T7700) serves as the basis.

Algorithm	Reference	Processor type	Frequency (GHz)	Factor
LMILS	-	Intel Core T7700	2.4	1.0
CPLEX (B&B)	[3]	Intel Xeon X7550	2.0	0.83
ITS	[3]	Intel Xeon X7550	2.0	0.83
Parallel ITS	[4]	Intel Xeon E5-2697	2.7	1.125

We set the stopping conditions of our algorithm according to the scaling factors given in Table 2 to make sure that our stopping conditions for different categories of instances are comparable to the strictest conditions. Following the ITS algorithm in [3], LMILS was performed 5 times on each instance under the following stopping conditions: 3000 seconds for A5, C5 and D3, 6000 seconds for D4 and D5, 1500 seconds for the remaining instances of the first and second categories, and 30000 seconds for all instances of the third category. The time limit of Parallel ITS [4] for the first and second categories was 1800 seconds longer than that of ITS. For the third category, the cutoff time of Parallel ITS and ITS was the same. The results from the exact CPLEX solver (B&B) were obtained by running the solver for a very long computation time (up to 9 days per instance). Table 3 summarizes the stopping conditions (cutoff time limit for each instance category) for the compared algorithms.

Table 3
Summary of stopping conditions for LMILS and three reference algorithms

Category	Instance	CPLEX (B&B)	ITS	Parallel ITS	LMILS
First	A5	9 days	3600s	5400s	3000s
	Others	9 days	1800s	3600s	1500s
Second	C5, D3	9 days	3600s	5400s	3000s
	D4, D5	9 days	7200s	9000s	6000s
	Others	9 days	1800s	3600s	1500s
Third	All	9 days	36000s	36000s	30000s

3.3 Comparative study

We compared the results achieved by our LMILS algorithm with those of reference algorithms on the three categories of instances.

3.3.1 Comparison on the instances of first-category

Table 4 provides the information of solution quality and computing time of LMILS and reference algorithms on the first-category instances. In columns 1 and 2 are their names and BKVs. Columns 3 and 4 present the best lower bound (LB) of each instance obtained by CPLEX (B&B) [3] and the computation time. The next 9 columns respectively report the best result f_{best}, average result f_{avg}, and average time $t_{avg}(s)$ to obtain the best results of ITS, Parallel ITS and LMILS over 5 runs. Additionally, the row "Avg." shows the average value of each column, and the row "#Best" indicates the number of instances for which an algorithm obtains the best values in term of f_{avg} or f_{best} among all the compared algorithms. The p-values from the non-parametric Friedman tests were introduced in the final row to check significant differences between LMILS and each reference algorithm in terms of the f_{best} and the f_{avg}. A p-value smaller than 0.05 means a significant difference. Finally, the best f_{best} and f_{avg} values among the compared results are indicated in bold and the optimal values are indicated by asterisks "*".

Table 4
Comparison between LMILS and reference algorithms on the first-category instances

Instance	BKV	CPLEX (B&B)		ITS			Parallel ITS			LMILS		
		LB	t	f_{best}	f_{avg}	$t_{avg}(s)$	f_{best}	f_{avg}	$t_{avg}(s)$	f_{best}	f_{avg}	$t_{avg}(s)$
A1	150,386*	**150,386**	5.3(h)	**150,386**	150,245.6	132.5	**150,386**	150,378.8	442.1	**150,386**	**150,386.0**	223.2
A2	179,973*	**179,973**	375(s)	**179,973**	**179,973.0**	101.7	**179,973**	**179,973.0**	1,545.8	**179,973**	**179,973.0**	26.3
A3	155,266	155,242	3.8(d)	**155,266**	154,909.2	515.7	**155,266**	**155,266.0**	707.6	**155,266**	**155,266.0**	106.8
A4	156,558	156,558	2.4(d)	156,423	156,338.2	400.2	156,540	156,453.0	2,675.3	**156,559**	**156,546.8**	496.1
A5	160,767	158,550	7.1(d)	160,276	159,963.0	951.2	160,767	160,438.8	3,991.5	**160,786**	**160,543.6**	1,939.4
B1	152,335	152,222	2.7(d)	**152,335**	152,290.0	66.3	**152,335**	152,306.2	334.1	**152,335**	**152,335.0**	65.4
B2	155,752	155,310	2.3(d)	155,554	155,526.2	231.8	**155,752**	155,593.6	1,625.6	**155,752**	**155,633.2**	551.1
B3	158,176	156,509	2.7(d)	157,551	157,455.0	753.8	158,176	157,556.0	2,632.5	**158,197**	**157,923.6**	667.3
B4	158,411	156,851	1.1(d)	158,403	158,215.0	819.7	158,411	158,254.2	3,287.3	**158,488**	**158,352.2**	899.8
B5	159,581	158,903	6.0(d)	159,426	158,827.0	675.7	159,581	158,966.4	1,841.6	**160,168**	**159,718.0**	1,021.4
Avg.	158,720.5	158,050.4	2.8(d)	158,559.3	158,374.2	464.9	158,718.7	158,518.6	1,908.3	**158,791.0**	**158,667.7**	599.7
#Best		2		4	1		5	2		**10**	**10**	
p-value		0.0047		0.0143	0.0027		0.0253	0.0047				

Table 4 reveals that LMILS performs remarkably on the 10 first-category instances. In terms of f_{best}, LMILS achieves all known optimal results (A1 and A2), and much better lower bounds for the remaining 8 instances compared with CPLEX (B&B) [3]. Besides, LMILS dominates both ITS (with 6 better and 4 equal results) and Parallel ITS (with 5 better and 5 equal results). LMILS significantly outperforms ITS and Parallel ITS in terms of f_{avg} by obtaining better or equal results on all instances. The small p-values (< 0.05) indicate that there are significant differences between our best results and those of ITS (p-value$= 0.0143$) and Parallel ITS (p-value$= 0.0253$).

3.3.2 Comparison on the instances of second-category

The comparative results of LMILS and reference algorithms on the second category are summarized in Table 5.

Table 5
Comparison between LMILS and reference algorithms on the second-category instances

Instance	BKV	CPLEX (B&B)		ITS			Parallel ITS			LMILS		
		LB	t	f_{best}	f_{avg}	$t_{avg}(s)$	f_{best}	f_{avg}	$t_{avg}(s)$	f_{best}	f_{avg}	$t_{avg}(s)$
C1	246,121*	**246,121**	1610.8(s)	**246,121**	**246,121.0**	8.7	**246,121**	**246,121.0**	68.6	**246,121**	**246,121.0**	1.0
C2	247,455*	**247,455**	5.3(h)	**247,455**	247,450.2	296.0	**247,455**	247,450.2	1,692.0	**247,455**	**247,455.0**	328.8
C3	249,070*	**249,070**	3.1(h)	**249,070**	**249,070.0**	78.0	**249,070**	**249,070.0**	1,227.4	**249,070**	**249,070.0**	31.1
C4	249,124*	**249,124**	19.7(h)	249,094	249,072.4	453.2	**249,124**	249,110.8	2,671.9	**249,124**	**249,118.0**	684.9
C5	249,935*	**249,935**	13.9(h)	249,881	249,756.8	1,469.2	**249,935**	249,881.2	3,163.5	**249,935**	**249,935.0**	925.9
D1	247,250	247,112	6.0(d)	247,199	247,170.6	487.7	**247,250**	247,185.4	1,434.4	**247,250**	**247,250.0**	273.4
D2	248,389	248,200	6.5(d)	248,389	247,968.6	820.7	248,389	248,087.4	1,748.3	**248,424**	**248,424.0**	474.3
D3	249,165	248,136	6.8(d)	248,923	248,663.8	2,056.8	249,165	248,818.4	3,677.6	**249,187**	**249,060.8**	1,735.1
D4	249,107	247,809	6.7(d)	249,107	248,672.2	4,480.0	248,949	248,691.0	8,164.1	**249,389**	**249,195.4**	2,794.0
D5	250,288	249,929	7.5(d)	250,158	249,956.8	2,866.3	250,288	250,019.2	8,453.3	**250,597**	**250,536.6**	3,823.9
Avg.	248,590.4	248,289.1	3.5(d)	248,539.7	248,390.2	1,301.7	248,574.6	248,443.5	3,230.1	**248,655.2**	**248,616.6**	1,107.2
#Best		5		3	2		6	2		**10**	**10**	
p-value		0.0253		0.0082	0.0047		0.0455	0.0047				

LMILS outperforms reference algorithms for all instances of the second category. Compared with B&B, LMILS easily finds all 5 known optimal results (C1, C2, C3, C4, C5) and improves the remaining 5 best results. Compared with ITS, LMILS obtains better results for 7 instances, and equal results for the other 3 instances. Compared with Parallel ITS, LMILS obtains better results for 4 instances, and equal results for the remaining 6 instances. Moreover, for same results, LMILS spends less time than reference algorithms to reach the BKVs (except for one instance C2), and reports better f_{avg} values for all instances. The p-values of 0.0082 and 0.0455 between LMILS and references algorithms (ITS, Parallel ITS) in terms of the f_{best} indicate that there are significant differences between their results.

3.3.3 Comparison on the instances of third-category

Table 6 lists the results on the third-category instances. It is worth mentioning that these instances have not been solved by CPLEX (B&B) [3].

Table 6
Comparison between LMILS and reference algorithms on the third-category instances

Instance	BKV	ITS			Parallel ITS			LMILS		
		f_{best}	f_{avg}	$t_{avg}(s)$	f_{best}	f_{avg}	$t_{avg}(s)$	f_{best}	f_{avg}	$t_{avg}(s)$
E1	623,606	619,800	618,852.0	10,672.3	**623,606**	621,962.0	19,931.6	**623,606**	**623,606.0**	8,319.8
E2	449,288	449,288	446,976.2	13,304.7	449,257	448,898.8	26,094.4	**452,622**	**452,286.4**	18,274.8
E3	551,531	548,110	545,262.2	12,292.0	551,531	549,323.0	32,071.5	**557,209**	**554,594.8**	15,978.8
E4	637,825	637,010	636,065.4	23,821.0	637,825	636,640.4	27,843.8	**639,875**	**639,440.6**	21,025.4
E5	474,222	472,158	470,955.4	23,995.0	474,222	471,652.8	31,226.6	**478,340**	**476,494.8**	17,544.9
F1	501,319	500,797	498,983.6	25,870.0	501,319	500,452.8	30,375.0	**505,729**	**505,194.2**	12,074.6
F2	495,887	494,333	490,600.4	27,228.5	495,887	494,511.6	33,638.6	**498,074**	**497,519.0**	21,244.8
F3	485,865	485,309	483,177.8	28,846.7	485,865	484,909.0	28,714.5	**488,811**	**488,300.8**	19,030.4
F4	485,126	484,835	481,769.8	21,927.3	485,126	484,126.4	20,739.4	**493,675**	**489,719.4**	23,055.4
F5	498,559	495,066	494,008.2	28,913.7	498,559	496,947.2	19,073.3	**503,416**	**501,358.6**	25,414.8
Avg.	520,322.8	518,670.6	516,665.1	21,687.1	520,320	518,942.4	26,970.9	**524,135.7**	**522,851.5**	18,196.4
#Best		0	0		1	0		**10**	**10**	
p-value		0.0016	0.0016		0.0027	0.0016				

Specifically, LMILS performs better than ITS for all instances, and better than Parallel ITS with 9 better and 1 equal results. In terms of f_{avg}, LMILS finds better results for all instances than ITS and Parallel ITS. The small p-values confirm the significant dominance of LMILS over reference algorithms. These results demonstrate the superiority of LMILS over each reference algorithm in terms of the best or/and average results.

To sum up, LMILS updates the best-known results for 18 instances among the 30 instances (60%) and matches the best-known results for the other cases, including 7 known optimal results.

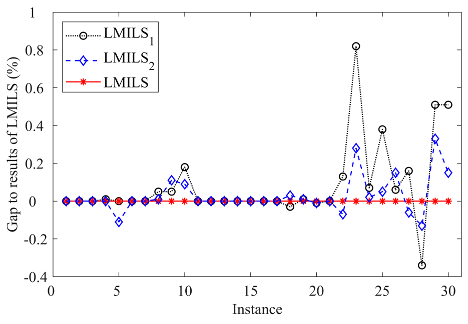
4 Analysis

The essential components of LMILS are the two learning-based mechanisms, the swap-based descent search, the incremental evaluation technique, and the intensified search mechanism. We analyze here their influences on the performance of LMILS. Following analysis adopted the same cutoff time limits as above.

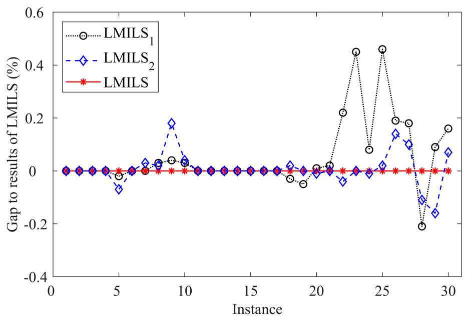
4.1 Effectiveness of two learning-based mechanisms

As explained in Sections 2.2 and 2.3.2, LMILS jointly applies learning mechanisms in the solution construction procedure and the perturbation procedure.

To justify their merits, two variants of LMILS, called LMILS$_1$ and LMILS$_2$, are created for comparison. LMILS$_1$ randomly constructs initial solutions without using the historical information vector η and LMILS$_2$ randomly selects a perturbation operator without the perturbation probability vector γ. Figure 4 shows their best results f_{best} and average results f_{avg} (over 5 runs). The X-axis indicates the instances that are numbered according to the order they appear in Tables 4-6. The Y-axis indicates the gap between f_{best} or f_{avg} and the new lower bound NLB from LMILS, i.e., $gap = \frac{NLB - f_{best}}{NLB} \times 100\%$ for Figure 4(a) and $gap = \frac{NLB - f_{avg}}{NLB} \times 100\%$ for Figure 4(b).



(a) Best results (f_{best})



(b) Average results (f_{avg})

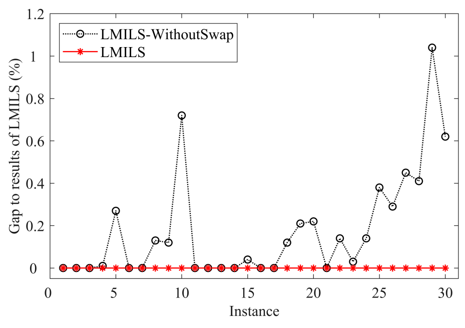
Fig. 4. Comparisons of LMILS (in red) with its variants LMILS$_1$ (in black) and LMILS$_2$ (in blue)

The best results in Figure 4(a) justify the significance of the two learning-based mechanisms in LMILS by 12 better, 15 equal and 3 worse results against LMILS$_1$, and 11 better, 14 equal and 5 worse results against LMILS$_2$. The average results in Figure 4(b) also demonstrate the advantage of the two learning-based mechanisms by 13 better, 13 equal and 4 worse results against LMILS$_1$, and 9 better, 15 equal and 6 worse results against LMILS$_2$.

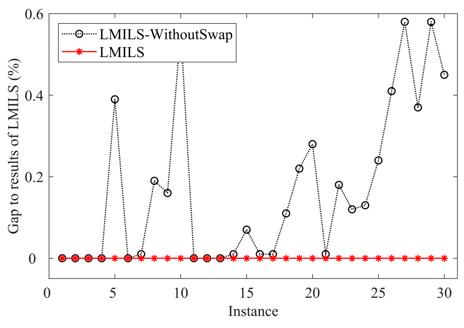
4.2 Impacts of the swap-based descent search

As described in Section 2.3.1, LMILS employs a two-phase local search, i.e., flip-based tabu search and swap-based descent search. To ensure a meaningful contribution of the swap descent search, we compare the performance of the modified LMILS without swap-based descent search (i.e., LMILS-WithoutSwap) and the normal LMILS on all 30 instances under the same experiment conditions as in Section 3.2. Figure 5 provides the results of their best and average performance (over 5 runs).

Figure 5 shows that LMILS obtains better f_{best} values for 18 instances, and better f_{avg} values for 22 instances compared with LMILS-WithoutSwap. Besides, removing the swap-based descent search deteriorates the performance



(a) Best results (f_{best})



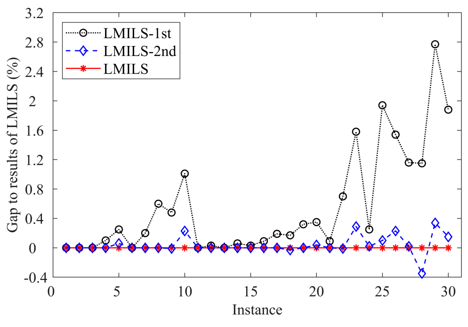
(b) Average results (f_{avg})

Fig. 5. Comparisons of LMILS with its weakened version that excludes swap-based descent search

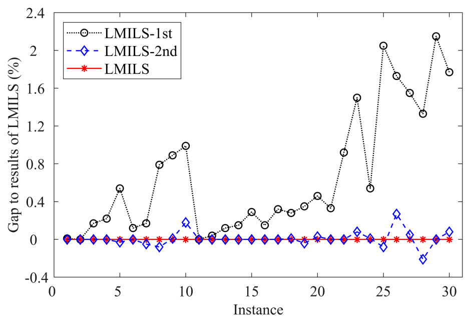
of LMILS for relatively difficult instances of each category, meaning that the swap-based descent search is essential for solving difficult instances.

4.3 Impacts of the incremental evaluation

To assess the usefulness of the fast incremental evaluation technique for the flip and swap operators, we compare LMILS with two variants (LMILS-1st and LMILS-2nd) by respectively 1) disabling the fast evaluation strategy of the flip operator; 2) disabling the fast evaluation strategy of the swap operator.



(a) Best results (f_{best})



(b) Average results (f_{avg})

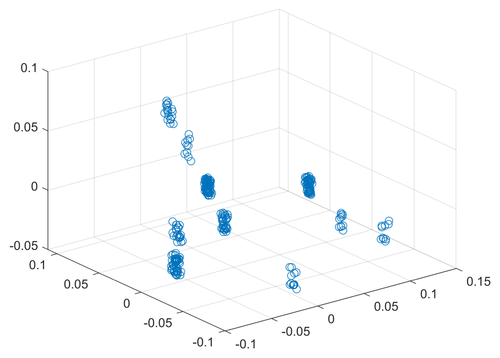
Fig. 6. Comparisons of LMILS (in red) with its variants LMILS-1st (in black) and LMILS-2nd (in blue)

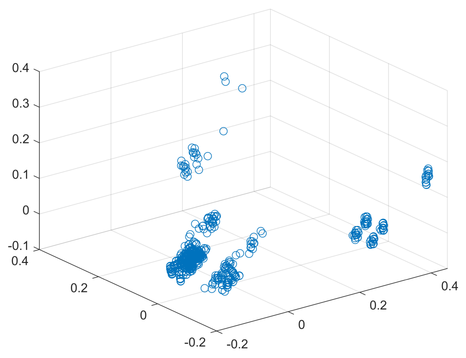
Figure 6 exhibits the best and average results (over 5 runs) of LMILS as compared with LMILS-1st and LMILS-2nd. Specifically, in terms of f_{best}, LMILS dominates LMILS-1st by obtaining 24 better and 6 equal results, and beats LMILS-2nd by obtaining 10 better, 16 equal, and 4 worse results. As for f_{avg}, LMILS performs better than LMILS-1st for 28 instances and LMILS-2nd for 9 instances. This indicates that these two fast evaluation strategies are key elements that ensure the high performance of LMILS. Moreover, LMILS-2nd outperforms LMILS-1st by reaching 24 better results in terms of f_{best}, and 28 better results in terms of f_{avg}. This implies that the fast evaluation strategy for the flip operator is more important than that for the swap operator. This is reasonable because the number of iterations of the flip-based tabu search is much larger than that of the swap-based descent search.

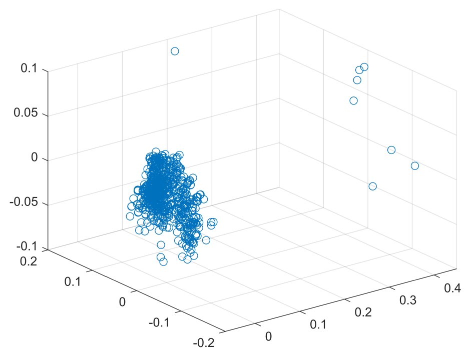
4.4 Cartography of search space

To know better the necessity of the intensified search mechanism (Section 2.3), this section conducts a study on the spatial distribution of high-quality solutions as in [13,21]. Our LMILS algorithm was run 10 times on each of 4 random instances (B5, D1, E1, and F5), and all local optimal solutions of high quality visited by LMILS were collected to characterize their spatial distribution. Here, a solution X with an objective value $f(X)$ larger than $0.999 \times NLB$ ($f(X) > 0.999 \times NLB$) is considered to be of high-quality. Following [13,21], we adopted the multidimensional scaling (MDS) method to approximately present the spatial distribution of high-quality solutions in the Euclidean space R^3 as follows. First, we calculated a distance matrix $D_{l \times l}$, where l is the number of local optimal solutions, and $d_{ij} \in D_{l \times l}$ is the distance between solutions X_i and X_j given by $d_{ij} = \frac{|X_i|+|X_j|-2|X_i \cap X_j|}{|X_i|+|X_j|}$. Then, according to the distance matrix obtained, we used the *cmdscale* method to obtain l coordinate points in the R^3 space to minimize the distortion of distance among the obtained coordinate points. In the end, the scatter graph of the resulting points was plotted in R^3.

Figure 7 shows the spatial distribution of the collected high-quality solutions visited by LMILS for the four selected instances. The high-quality solutions are clustered respectively within a sphere of small diameters. This implies that high-quality neighboring solutions can be found around a discovered high-quality solution. Thus, LMILS systematically starts the perturbation procedure from each best solution X_{best} ever found in the IILS procedure to discover other high-quality solutions. Second, the clusters are far separated. Thus, to move from a promising area (i.e., a cluster of high-quality solutions) to a new promising area (a different cluster of high-quality solutions), it is indispensable for the algorithm to jump far and perform strong diversification. In LMILS, this is achieved by the learning-driven solution construction procedure and perturbation procedure. This experiment provides highly relevant supporting information for the design of our algorithm.


(a) Distribution of 283 local optima for B5


(b) Distribution of 538 local optima for D1


(c) Distribution of 772 local optima for E1

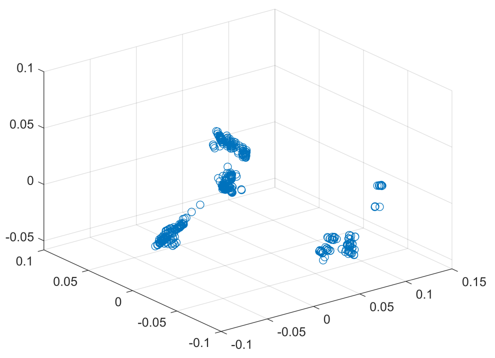
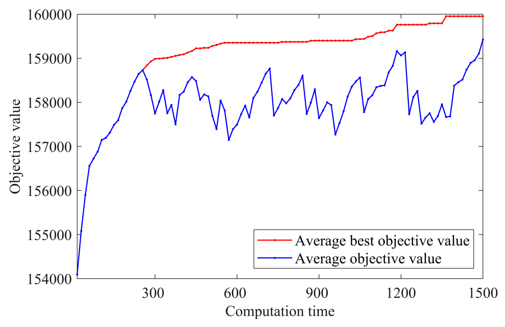

(d) Distribution of 1325 local optima for F5

Fig. 7. Distribution of the high-quality local optima for four instances

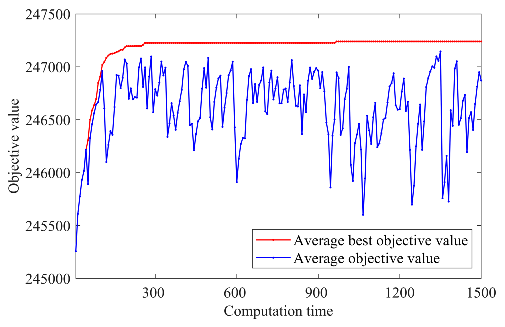
4.5 Convergence analysis

This section is dedicated to a convergence analysis of the LMILS algorithm by studying the convergence profiles of the algorithm on 4 representative instances (B5, D1, E1, F5). The experiment for this study follows the same experimental and stopping conditions of Section 3.2. Figure 8 shows the convergence profiles of LMILS where the X-axis indicates the computation time in seconds and the Y-axis indicates the objective values (red curves represent the average best objective values and blue curves represent the average current objective values over 5 independent runs, respectively).

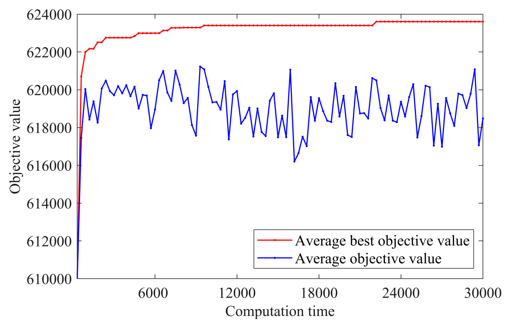
From Figure 8, we can make the following comments. At the beginning of the search, the solution quality of the algorithm improves dramatically and rapidly. Then, as the quality improvement slows down, the algorithm gradually converges to its best solution. For all four instances (B5, D1, E1 and F5), the quality fluctuations of the current solutions are important throughout the search process. This behavior is beneficial to the algorithm, as it allows the algorithm to escape many local optimal solutions. This experiment is a



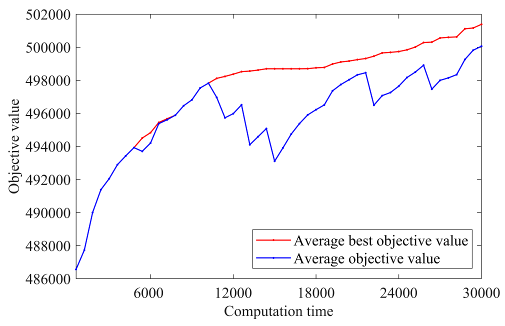
(a) B5



(b) D1



(c) E1



(d) F5

Fig. 8. Convergence profiles of LMILS on four instances (B5, D1, E1, F5)

relevant indicator for the explanation of the good performance of the LMILS algorithm.

5 Conclusions

The profit maximization set covering problem (PMSCP) is a useful model for a number of applications. However, solving the problem remains a difficult task. We presented in this work a learning-based multi-start iterated local search to solve this computationally challenging problem. The algorithm features several complementary search components including an intensification-driven local search to perform effective examination of candidate solution exploration, a learning-driven construction to obtain promising starting solutions, and a learning-based perturbation to select a proper perturbation.

We verified the performance of the proposed algorithm on 30 instances against the state-of-the-art methods. The comparative study revealed that our algorithm performs remarkably well by discovering 18 new lower bounds and reaching the current best results for the 12 rest instances, including 7 known optimal values. Besides, the benefits of the learning-based strategy and the intensification-driven local search were also investigated.

Future studies can be conducted at following directions. First, this work uses simple learning techniques for initial solution construction and perturbation selection. To go further, new search algorithms driven by other learning techniques such as reinforcement/probability learning [30,15] and opposition-based learning [31] can be researched. Second, other general search frameworks can be adapted to solve the PMSCP, including frequent pattern-based search [29], fixed set search [11,10] and memetic search [19]. The algorithm presented in this work can be beneficially used as a key intensification component. Third, the basic ideas of the proposed approach are rather general. As such, it would be interesting to test these ideas on other related problems, such as the clustered set covering problem [1], the maximum group set coverage problem [8], and budgeted maximum coverage problem [15]. Finally, as far as the exact solution of the problem is concerned, only the general CPLEX solver (B&B) has been studied in the literature. It would be interesting to design dedicated exact algorithms capable of exploring special features of the profit maximization set covering problem.

6 Acknowledgments

We are grateful to the reviewers for their helpful comments. This work was partially supported by the National Natural Science Foundation of China [Grant No. 72122006, No. 62101125], Natural Science Foundation of Jiangsu Province [Grant No. SBK2020040023], and Fundamental Research Funds for the Central Universities [Grant No. 2242022R40067, No. 2242022K30007].

References

[1] L. Alfandari, J. Monnot, A note on the clustered set covering problem, Discrete Applied Mathematics 164 (2014) 13–19.

[2] J. E. Beasley, P. C. Chu, A genetic algorithm for the set covering problem, European Journal of Operational Research 94 (2) (1996) 392–404.

[3] N. Bilal, P. Galinier, F. Guibault, An iterated-tabu-search heuristic for a variant of the partial set covering problem, Journal of Heuristics 20 (2) (2014) 143–164.

[4] A. F. Brum, O. C. B. de Araújo, F. M. Müller, Uma metaheurıstica hıbrida para o problema de recobrimento de conjuntos com agrupamento, in: Proceedings of the Brazilian Symposium on Operations Research, 2015, pp. 2727–2738.

[5] Y. N. Chen, J. K. Hao, An iterated "hyperplane exploration" approach for the quadratic knapsack problem, Computers & Operations Research 77 (2017) 226–239.

[6] B. Crawford, R. Soto, F. Cisternas-Caneo, D. Tapia, H. de la Fuente Mella, W. Palma, J. Lemus-Romani, M. Castillo, M. Becerra-Rozas, A comparison of learnheuristics using different reward functions to solve the set covering problem, in: B. Dorronsoro, L. Amodeo, M. Pavone, P. Ruiz (Eds.), Optimization and Learning - 4th International Conference, 2021, pp. 74–85.

[7] M. Dorigo, V. Maniezzo, A. Colorni, Ant system: optimization by a colony of cooperating agents, IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) 26 (1) (1996) 29–41.

[8] H. W. Du, Z. Zhang, Z. H. Duan, C. Tian, D. Z. Du, Formulate full view camera sensor coverage by using group set coverage, in: Z. J. Haas, R. Prakash, H. Ammari, W. L. Wu (Eds.), Wireless Internet - 15th European Alliance for Innovation International Conference, Springer, 2022, pp. 76–90.

[9] F. W. Glover, M. Laguna, Tabu search, in: Handbook of Combinatorial Optimization, Springer, Boston, 1998, pp. 2093–2229.

[10] R. Jovanovic, A. Sanfilippo, S. Voß, Fixed set search applied to the multi-objective minimum weighted vertex cover problem, Journal of Heuristics 28 (4) (2022) 481–508.

[11] R. Jovanovic, S. Voß, Fixed set search application for minimizing the makespan on unrelated parallel machines with sequence-dependent setup times, Applied Soft Computing 110 (2021) 107521.

[12] R. M. Karp, Reducibility among combinatorial problems, in: R. E. Miller, J. W. Thatcher, J. D. Bohlinger (Eds.), Proceedings of a Symposium on the Complexity of Computer Computations, Plenum Press, New York, 1972, pp. 85–103.

[13] X. J. Lai, J. K. Hao, Iterated maxima search for the maximally diverse grouping problem, European Journal of Operational Research 254 (3) (2016) 780–800.

[14] X. J. Lai, Z. P. Lü, Multistart iterated tabu search for bandwidth coloring problem, Computers & Operations Research 40 (5) (2013) 1401–1409.

[15] L. W. Li, Z. Q. Wei, J. K. Hao, K. He, Probability learning based tabu search for the budgeted maximum coverage problem, Expert Systems with Applications 183 (2021) 115310.

[16] Y. L. Lu, J. K. Hao, Q. H. Wu, Hybrid evolutionary search for the traveling repairman problem with profits, Information Sciences 502 (2019) 91–108.

[17] R. Martí, A. Duarte, M. Laguna, Advanced scatter search for the max-cut problem, INFORMS Journal on Computing 21 (1) (2009) 26–38.

[18] R. Martí, J. M. Moreno-Vega, A. Duarte, Advanced multi-start methods, in: Handbook of Metaheuristics, Springer, Boston, 2010, pp. 265–281.

[19] P. Moscato. Memetic algorithms: A short introduction, in: D. Corne et al. (Eds) New Ideas in Optimization, McGraw-Hill Ltd., UK, 1999, pp. 219–234.

[20] K. S. Narendra, M. A. L. Thathachar, Learning automata - an introduction, Prentice Hall, 1989.

[21] D. C. Porumbel, J. K. Hao, P. Kuntz, A search space “cartography” for guiding graph coloring heuristics, Computers & Operations Research 37 (4) (2010) 769–778.

[22] G. Schrimpf, J. Schneider, H. Stamm-Wilbrandt, G. Dueck, Record breaking optimization results using the ruin and recreate principle, Journal of Computational Physics 159 (2) (2000) 139–171.

[23] T. Stützle, H. H. Hoos, MAX-MIN ant system, Future Generation Computer Systems 16 (8) (2000) 889–914.

[24] W. Sun, J. K. Hao, W. L. Li, Q. H. Wu, An adaptive memetic algorithm for the bidirectional loop layout problem, Knowledge-Based Systems 258 (2022) 110002.

[25] F. L. Usberti, P. M. França, A. L. M. França, GRASP with evolutionary path-relinking for the capacitated arc routing problem, Computers & Operations Research 40 (12) (2013) 3206–3217.

[26] Z. Q. Wei, J. K. Hao, Multistart solution-based tabu search for the set-union knapsack problem, Applied Soft Computing 105 (2021) 107260.

[27] Q. H. Wu, J. K. Hao, An adaptive multistart tabu search approach to solve the maximum clique problem, Journal of Combinatorial Optimization 26 (1) (2013) 86–108.

[28] T. Q. Zhou, Z. P. Lü, Y. Wang, J. W. Ding, B. Peng, Multi-start iterated tabu search for the minimum weight vertex cover problem, Journal of Combinatorial Optimization 32 (2) (2016) 368–384.

[29] Y. M. Zhou, J. K. Hao, B. Duval, Frequent pattern-based search: a case study on the quadratic assignment problem. IEEE Transactions on Systems, Man, and Cybernetics: Systems 52(3) (2022) 1503–1515.

[30] Y. M. Zhou, J. K. Hao, B. Duval, Reinforcement learning based local search for grouping problems: A case study on graph coloring. Expert Systems with Applications 64 (2016) 412–422.

[31] Y. M. Zhou, J. K. Hao, B. Duval, Opposition-based memetic search for the maximum diversity problem. IEEE Transactions on Evolutionary Compututation 21(5) (2017) 731–745.

A Importance of the reduction procedure

The reduction procedure described in Section 2.1 is a significant component of the LMILS algorithm. To demonstrate its effectiveness, we additionally compared the LMILS algorithm with the LMILS_DR, a LMILS variant in which the reduction procedure is disabled. We ran both algorithms 5 times on each of 30 instances under the same experimental conditions as in the paper. The results are summarized in Table A.1. Column 1 shows the name of instances, columns 2-3 respectively report the number of subsets before and after the reduction procedure. The next 6 columns indicate the detailed results (f_{best}, f_{avg} and t_{avg}) of LMILS and LMILS_DR. The row Avg. shows the average value for each column.

Table A.1
Comparison between LMILS and LMILS_DR on all the 30 instances

Instance	Number of subsets $\|S\|$	Number of reduced subsets $\|S_0\|$	LMILS_DR			LMILS		
			f_{best}	f_{avg}	$t_{avg}(s)$	f_{best}	f_{avg}	$t_{avg}(s)$
A1	3,493	3,212	**150,386**	**150,386.0**	411.2	**150,386**	**150,386.0**	223.2
A2	4,308	4,060	**179,973**	**179,973.0**	175.7	**179,973**	**179,973.0**	26.3
A3	6,657	6,015	**155,266**	**155,266.0**	54.0	**155,266**	**155,266.0**	106.8
A4	13,076	11,900	**156,559**	**156,547.4**	480.1	**156,559**	156,546.8	496.1
A5	82,635	76,932	**160,845**	**160,626.2**	1,097.8	160,786	160,543.6	1,939.4
B1	6,924	6,362	**152,335**	**152,335.0**	110.4	**152,335**	**152,335.0**	65.4
B2	10,617	9,638	**155,752**	**155,672.8**	554.4	**155,752**	155,633.2	551.1
B3	26,043	23,654	158,176	**157,944.6**	645.8	**158,197**	157,923.6	667.3
B4	40,351	36,601	158,429	158,303.6	814.3	**158,488**	**158,352.2**	899.8
B5	70,899	64,087	159,811	159,475.6	856.8	**160,168**	**159,718.0**	1,021.4
C1	6,633	5,891	**246,121**	**246,121.0**	1.2	**246,121**	**246,121.0**	1.0
C2	12,088	10,651	**247,455**	**247,455.0**	226.4	**247,455**	**247,455.0**	328.8
C3	24,007	21,312	**249,070**	**249,070.0**	43.2	**249,070**	**249,070.0**	31.1
C4	36,689	32,516	**249,124**	249,112.0	491.5	**249,124**	**249,118.0**	684.9
C5	63,635	56,449	**249,935**	**249,935.0**	814.3	**249,935**	**249,935.0**	925.9
D1	15,252	13,303	**247,250**	247,247.6	492.7	**247,250**	**247,250.0**	273.4
D2	28,303	24,694	**248,424**	**248,424.0**	283.0	**248,424**	**248,424.0**	474.3
D3	55,975	49,034	**249,269**	**249,073.8**	2,146.6	249,187	249,060.8	1,735.1
D4	85,978	75,264	249,354	**249,227.8**	2,793.3	**249,389**	249,195.4	2,794.0
D5	117,274	103,760	**250,617**	250,484.0	2,375.4	250,597	**250,536.6**	3,823.9
E1	10,325	8,521	**623,606**	623,464.8	18,998.5	**623,606**	**623,606.0**	8,319.8
E2	23,215	16,264	451,416	450,878.2	13,251.9	**452,622**	**452,286.4**	18,274.8
E3	38,805	30,673	554,885	553,098.6	17,137.5	**557,209**	**554,594.8**	15,978.8
E4	61,207	57,508	639,863	639,434.0	15,156.6	**639,875**	**639,440.6**	21,025.4
E5	78,331	67,368	475,706	475,183.4	16,964.6	**478,340**	**476,494.8**	17,544.9
F1	94,685	86,308	505,716	504,329.0	19,060.8	**505,729**	**505,194.2**	12,074.6
F2	122,973	105,959	**498,173**	497,231.4	22,499.1	498,074	**497,519.0**	21,244.8
F3	167,911	148,104	**490,554**	**489,111.0**	15,717.7	488,811	488,300.8	19,030.4
F4	276,521	239,350	490,289	488,855.6	20,125.3	**493,675**	**489,719.4**	23,055.4
F5	462,666	408,734	501,541	500,755.6	18,170.9	**503,416**	**501,358.6**	25,414.8
Avg.	68,249.2	60,137.5	310,196.7	309,834.1	6,398.4	**310,527.3**	**310,045.3**	6,634.4

One observes from Table A.1 that LMILS obtained better results than LMILS_DR for 11 instances in terms of f_{best} and for 14 instances in terms of f_{avg}. This proved that the reduction procedure adopted in this study plays an important role for the high performance of the LMILS algorithm.