Iterated hyperplane search for the budgeted maximum coverage problem

Zequn Wei[a], Jin-Kao Hao[b,*]

[a]School of Economics and Management, Beijing University of Posts and Telecommunications, 100876 Beijing, China
[b]LERIA, Université d'Angers, 2 Boulevard Lavoisier, 49045 Angers, France

Expert Systems with Applications, https://doi.org/10.1016/j.eswa.2022.119078, Oct 2022

ARTICLE INFO

Keywords:
Budgeted maximum coverage
Knapsack
Heuristics
Iterated local search
Hyperplane search
Combinatorial optimization

ABSTRACT

We present an iterated hyperplane search approach for the budgeted maximum coverage problem. Our algorithm relies on the idea of searching on specific areas identified by cardinality-constrained hyperplanes. It combines three complementary procedures: a tabu search procedure to identify a promising hyperplane, a hyperplane search procedure to examine candidate solutions on cardinality-constrained hyperplanes and a dedicated perturbation procedure to ensure the diversification of the search. We show the competitiveness of the algorithm on 30 benchmark instances and present experiments to study the key components of the algorithm.

1. Introduction

The maximum coverage problem (MCP) (Hochba, 1997) and its variants are general models with interesting applications (Chauhan, Unnikrishnan and Figliozzi, 2019; Kartashov, SergiyYakoIlev, YakoIleva and Shekhovtsov, 2019; Liang, Shen and Chen, 2021). The budgeted maximum coverage problem (BMCP) generalizes the MCP. Given a set $E = \{1, \dots, n\}$ of n elements and a set $\mathcal{I} = \{1, \dots, m\}$ of m items, where each element has a positive gain (or profit) and each item is a subset of E with a positive weight. The relationships between the items of $\mathcal{I}$ and the elements of E are given by a $m \times n$ binary matrix M. Then BMCP aims to find a subset $\mathcal{S} \subseteq \mathcal{I}$ to maximize the profit sum $P(\mathcal{S})$ of the covered elements while ensuring that the weight sum $W(\mathcal{S})$ of the selected items satisfies a given budget (knapsack capacity) C (Khuller, Moss and Naor, 1999). Formally, BMCP can be written as follows.

$$(BMCP) \qquad \text{Maximize} \quad P(\mathcal{S}) = \sum_{j \in E_{\mathcal{S}}} p_j \tag{1}$$

$$\text{subject to} \quad W(\mathcal{S}) = \sum_{i \in \mathcal{S}} w_i \leqslant C \tag{2}$$

$$\mathcal{S} \subseteq \mathcal{I}, \ \ E_{\mathcal{S}} = \cup_{i \in \mathcal{S}} E_i \tag{3}$$

with $\mathcal{S} \subseteq \mathcal{I}$ being the selected item set and $E_{\mathcal{S}} \subseteq E$ being the elements covered by the item set $\mathcal{S}$. It is worth noting that the profit of an element will be calculated once in Equation (1), even if it appears in multiple selected items.

BMCP can model numerous real-life applications. We take the case in the area of flexible production networks (or supply chain networks with flexible plants). Suppose that a company has a number of plants that produce different types of commodities and some commodities can be produced by different plants. Each plant has a cost (e.g., representing production and operational expenses) and each produced commodity generates a profit. Usually, the company uses the process flexibility strategy (Simchi-Levi, 2010; Cachon and Terwiesch, 2011) to satisfy the variability of product demands. So the company needs to decide which plants are to be used for production to maximize the sum of profit of the produced commodities while ensuring the sum of budget required by the selected plants satisfies the given budget. This problem is equivalent to the BMCP model when an item represents a plant with its cost and an element represents a commodity with its profit. Moreover, BMCP has other practical applications related to the locations of network monitors (Suh, Guo, Kurose and Towsley, 2006), news recommendation (Li, Wang, Li, Knox and Padmanabhan, 2011), software defined networking (Kar, Wu and Lin, 2016), text summarization (Takamura and Okumura, 2009), and program assignment (Li, Wei, Hao and He, 2021).

Depending on the optimization objective and constraints, several decision-making problems related to BMCP have been proposed and studied, such as the maximum coverage problem with group budget constraints (Chekuri and Kumar, 2004), the generalized maximum coverage problem (Cohen and Katzir, 2008), the budgeted maximum coverage with overlapping costs (Curtis, Pemmaraju and Polgreen, 2010), the set-union knapsack (Goldschmidt, Nehme and Yu, 1994) (SUKP), etc. These problems as well as BMCP are difficult to solve due to their $\mathcal{NP}$-hard nature. Among these problems, SUKP and BMCP can be regarded as a pair of "dual" problems (Li et al., 2021). Several solution approaches have been

*Corresponding author
Email addresses: zequn.wei@gmail.com (Z. Wei); jin-kao.hao@univ-angers.fr (J. Hao)
ORCID(s):

proposed for solving for SUKP, including: exact algorithm (Goldschmidt et al., 1994), approximation algorithm (Taylor, 2016), population-based hybrid algorithms (He, Xie, Wong and Wang, 2018; Ozsoydan and Baykasoglu, 2019; Liu and He, 2019) and local search algorithms (Lin, Guan, Li and Feng, 2019; Wei and Hao, 2021).

However, for BMCP, only two algorithms have been devised in the literature. First, Khuller et al. (Khuller et al., 1999) introduced an approximation algorithm, which has a guaranteed approximation ratio of $(1-1/e)$. Recently, Li et al. (Li et al., 2021) proposed the first probability learning based tabu search (PLTS) approach, which integrates the reinforcement learning technique and local optimization. Based on 30 benchmark instances (see Section 3.1), they demonstrated the superiority of PLTS compared with the reference algorithms they tested. They also reported lower and upper bounds for these 30 instances with the CPLEX solver under the running time of 5 hours.

In this paper, we enrich the toolbox for practically solving BMCP by proposing an iterated hyperplane search algorithm (IHS) with the following contributions. First, the proposed iterated hyperplane search algorithm (IHS) is the first BMCP algorithm, which explores the idea of searching on specific areas identified by cardinality-constrained hyperplanes. During the search process, a promising hyperplane is identified by a local search procedure at first, then a focused hyperplane search is carried out to examine in detail this hyperplane as well as some nearby hyperplanes. To ensure a suitable diversification of the search process, the algorithm additionally integrates a dedicated perturbation procedure to complement the intensification-oriented hyperplane search. Second, the comparative study on the 30 benchmark instances indicates that our algorithm is highly effective and robust compared to the best performing BMCP algorithms. Our algorithm discovers new lower bounds for 18 instances, while requiring a short running time (typically less than one minute) to attain its best solutions. We make our algorithm publicly available to facilitate its use in future research on BMCP and for solving related practical problems.

2. Iterated hyperplane search for the BMCP

The basic idea of our IHS approach is to explore efficiently promising sub-spaces ("hyperplanes") around attained high-quality local optima. This section starts with some basic notations and definitions involved in our approach and then describes its main components.

2.1. Notations and definitions

By reference to the item set $\mathcal{I} = \{1, 2, \ldots, m\}$, we represent a solution of BMCP by $S = (x_1, \ldots, x_m)$, where $x_i = 1$ if item i is selected, $x_i = 0$ otherwise. Then we define the search space Ω of the given instance as the collection of all non-empty subsets of $\mathcal{I}$, i.e.,

$$\Omega = \{x : x \in \{0,1\}^m\} \tag{4}$$

We define the feasible search space Ω^F as the set of all feasible solutions satisfying the given budget C, i.e.,

$$\Omega^F = \{x \in \{0,1\}^m : \sum_{i=1}^{m} w_i x_i \leqslant C\} \tag{5}$$

Let $S = (x_1, \ldots, x_m)$ be a solution, let k denote the number of selected items in $S = (x_1, \ldots, x_m)$, i.e., $k = \sum_{i=1}^{m} x_i$. We define the restricted subspace $\Omega^{[k]}$ as a subspace of Ω such that each solution satisfies the k-dimensional hyperplane constraint.

$$\Omega^{[k]} = \{x \in \{0,1\}^m : \sum_{i=1}^{m} x_i = k\} \tag{6}$$

Note that each k-dimensional hyperplane $\Omega^{[k]}$ includes both feasible and infeasible S. It is clear that $\Omega = \cup_{k=1}^{m} \Omega^{[k]}$.

Given a solution S and its total weight $W(S) = \sum_{i \in S} w_i$, its *overweight* is defined as follows.

$$W_o(S) = max\{W(S) - C, 0\} \tag{7}$$

where C is the given budget.

2.2. Main framework

The proposed IHS algorithm focuses on exploring some promising subspaces identified by cardinality-constrained hyperplanes, instead of searching in the whole space. The idea of using hyperplanes to limit the search scope has been successfully employed for solving knapsack problems such as multidimensional knapsack (Fleszar and Hindi, 2009; Vasquez and Hao, 2001) and quadratic knapsack problem (Chen and Hao, 2017).

IHS combines three complementary procedures, i.e., a tabu search procedure to identify a promising hyperplane, a hyperplane search procedure to examine candidate solutions on cardinality-constrained subspaces and a dedicated perturbation procedure to diversify the search.

Specifically, as shown in Algorithm 1, the algorithm starts with a dynamic greedy initialization procedure (line 3, Alg. 1, and Section 2.3) to generate a feasible solution of reasonable quality in the space Ω. The algorithm first initializes the best solution S^* found so far (line 4, Alg. 1), then runs a "while" loop (lines 5-13, Alg. 1) to perform the main procedure until the cut-off limit t_{max} is met. At each "while" loop, the solution S is improved by the tabu search procedure with a union neighborhood $N_1 \cup N_2$ limited by search depth I_{max1} (line 6, Alg. 1, and Section 2.4). After determining the hyperplane dimension k from the best solution of tabu search (line 7, Alg. 1), the hyperplane search procedure uses the neighborhood N_3 to successively examine candidate solutions in the spaces $\Omega^{[k]}, \Omega^{[k]}, \ldots, \Omega^{[k+\gamma]}$ where γ is a parameter (line 8, Alg. 1). The overall best solution S^* will be updated conditionally (lines 9-11, Alg. 1), and then the solution from hyperplane search is modified by the greedy randomized perturbation procedure. The perturbed solution is employed as the input solution of the next "while" loop. Finally, IHS terminates when the cut-off time t_{max} is met while returning the recorded overall best solution S^*.

Algorithm 1 Iterated hyperplane search for BMCP

```
 1: Input: Instance $I$, cut-off time $t_{max}$, depth of tabu search $I_{max1}$, depth of
    hyperplane search $I_{max2}$, limit of hyperplane search $\gamma$, neighborhoods
    $N_1$ ($flip$), $N_2$ ($swap$), $N_3$ (swap with infeasible search space),
    perturbation strength $\delta$.
 2: Output: The overall best solution $S^*$ found.
 3: $S \leftarrow Dynamic\_Greedy\_Initialization(I)$ /* $S$ is the current solution
    */
 4: $S^* \leftarrow S$
 5: while $Time \le t_{max}$ do
 6:     $S \leftarrow Tabu\_Search(S, N_1 \cup N_2, I_{max1})$    /* Improve $S$ by tabu
        search */
 7:     $k \leftarrow Determine\_Hyperplane\_Dimension(S)$    /* Calculate the
        hyperplane dimension $k$ from the solution of tabu search */
 8:     $S \leftarrow Hyperplane\_Search(S, k, \gamma, N_3, I_{max2})$    /* Improve $S$ by
        hyperplane search starting from hyperplane $k$ */
 9:     if $f(S) > f(S^*)$ then
10:         $S^* \leftarrow S$
11:     end if
12:     $S \leftarrow Greedy\_Randomized\_Perturbation(S, \delta)$
13: end while
14: return $S^*$
```

2.3. Dynamic greedy initialization

Given a solution S and the set E_S of the covered elements, let u be a non-selected item ($u \notin S$) and E_u be the set of corresponding elements. Then, we define the *contribution* CT of the non-selected item u to the solution S by $CT_u = \sum_{j \in E_u \wedge j \notin E_S} p_j$ and the dynamic density DD of the item u by $DD_u = CT_u / w_u$ where w_u is the weight of the item u. It is clear that items with a high DD_u value are more interesting than those with a low DD_u value. The dynamic greedy initialization procedure uses the high DD_u items to create an initial solution.

The initialization procedure uses a "while" loop to iteratively add non-selected items into the initial solution S (see Algorithm 2). Specifically, we first calculate the contribution CT_u of each non-selected item u in the item set $\mathcal{I}$ and the associated dynamic density DD_u. Then, the item u^* with the maximum density is identified and added into S. This process is iterated until the budget C is met.

2.4. Tabu search

The IHS algorithm applies the well-known tabu search (TS) approach (Glover and Laguna, 1997) to explore local optima within the feasible space Ω^F. As shown in Algorithm 3, the TS procedure first initializes the tabu list and the local best solutions S_b (lines 4-5, Alg. 3). Then the main search process performs a "while" loop to examine candidate solutions in Ω^F (lines 6-17, Alg. 3). At each iteration, all admissible neighbor solutions in the union neighborhood $N_1(S) \cup N_2(S)$ are identified and the best admissible neighbor solution S' is used to update S (lines 7-11, Alg. 3, see Section 2.4.1). After each solution

Algorithm 2 Dynamic greedy initialization

```
 1: Input: Instance $I$.
 2: Output: A feasible solution $S$.
 3: $S \leftarrow \emptyset$
 4: $W(S) \leftarrow 0$
 5: while $W(S) \le C$ do
 6:     $CT \leftarrow Calculate\_contribution()$   /* Calculate the contribution of
        each non-selected item */
 7:     $DD \leftarrow Calculate\_dynamic\_density(CT)$         /* Calculate the
        dynamic density of each non-selected item */
 8:     Identify the item $u^*$ with the maximum density in $DD$
 9:     if $w(u^*) + W(S) \le C$ then
10:         $S \leftarrow Add\_item(u^*, S)$
11:     else
12:         break;
13:     end if
14: end while
15: return $S$
```

transition, the best solution S_b found so far and the tabu list are updated accordingly (lines 12-18, Alg. 3, see Section 2.4.2). When S_b cannot be improved for I_{max1} consecutive iterations, the TS procedure terminates and returns its best feasible solution S_b found.

Algorithm 3 Tabu search procedure

```
 1: Input: Input solution $S$, neighborhood $N_1$, $N_2$, the maximum number
    of iterations $I_{max1}$.
 2: Output: Best solution $S_b$ found during tabu search.
 3: Initialize $tabu\_list$
 4: $S_b \leftarrow S$
 5: $i \leftarrow 0$
 6: while $i \le I_{max1}$ do
 7:     Find all admissible neighbor solutions $N'(S)$ in $N_1(S) \cup N_2(S)$
 8:     if $N'(S) \neq \emptyset$ then
 9:         /* Select the best admissible neighbor solution $S'$ in $N'(S)$*/
            $S' \leftarrow argmax\{f(S') : S' \in N'(S)\}$
10:     end if
11:     $S \leftarrow S'$         /* The selected neighbor solution becomes the new
        current solution */
12:     if $f(S) > f(S_b)$ then
13:         $f(S_b) \leftarrow f(S)$
14:         $i \leftarrow 0$
15:     else
16:         $i \leftarrow i + 1$
17:     end if
18:     Update $tabu\_list$
19: end while
20: return $S_b$
```

2.4.1. *Neighborhood and its evaluation*

To explore efficiently the feasible space Ω^F, our TS procedure employs a constrained union neighborhood induced by two common move operators $flip$ and $swap$ (Li et al., 2021; Wei and Hao, 2021; Aïder, Gacem and Hifi, 2022; Avci and Topaloglu, 2017). The $flip(p)$ operator simply changes the value of a binary variable i of the current solution, implying that either a new item is added into the solution or a selected item is removed from the solution. The operation $swap(q, p)$ exchanges a selected item against a non-selected item to generate a new solution. Let $S' = S \oplus \{flip, swap\}$ be the feasible neighbor solution achieved by the move operators. Then the associated neighborhoods N_1 and N_2 induced by $flip$ and $swap$ can be written as follows, respectively.

$$N_1(S) = \{S' : S' = S \oplus flip(p), p \in S, \sum_{i=1}^{m} w_i x_i \leqslant C\} \quad (8)$$

$$N_2(S) = \{S' : S' = S \oplus swap(q,p), q \in V, p \in \bar{V}, \sum_{i=1}^{m} w_i x_i \leqslant C\} \quad (9)$$

where V and $\bar{V}$ denote the selected and non-selected item set, respectively.

The TS procedure explores the constrained union neighborhood $N_1 \cup N_2$. Given a neighbor solution S' in $N_1 \cup N_2$, the time complexity of calculating the total profit $P(S')$ is $O(mn)$. However, we can easily obtain the weight $W(S')$ of S' in $O(1)$, i.e., $W(S') = W(S) + w_p - w_q$. To speed up the examination of the neighborhood $N_1 \cup N_2$, we adopt a weight-based filtering strategy to avoid calculating the total profit of infeasible neighbor solutions. Specifically, at each iteration of TS(line 7, Alg. 3), we calculate the total weight $W(S')$ rather than the total profit $P(S')$ at first. Then, only feasible neighbor solutions S' that satisfy the budget constraint (i.e., $W(S') \leq C$) will be further considered. This filtering strategy considerably reduces the effort for evaluating the union neighborhood $N_1 \cup N_2$.

To further increase the computational efficiency when calculating the total profit $P(S')$, we adopt a dedicated $gain\_updating$ strategy inspired by a streamlining technique designed for SUKP (Lin et al., 2019; Wei and Hao, 2020). For each neighboring solution S', we employ an n-vector G to count the frequencies of elements in S'. In this way, we can quickly obtain $P(S')$ by checking only the elements whose value changes in vector G. Let $P(S)$ be the total profit of S. For each changed element j in S', the $gain\_updating$ strategy covers three different cases.

$$P(S') = \begin{cases} P(S) + p_j, & \text{if } G_j \text{ changes from zero to non-zero;} \\ P(S) - p_j, & \text{if } G_j \text{ changes from non-zero to zero;} \\ P(S), & \text{otherwise.} \end{cases} \quad (10)$$

Let $d_i = \sum_{j=1}^{n} M_{ij}$ be the number of elements covered by item i, where M is the binary relationship matrix of the given instance. Suppose that $d_{max} = max_{i=1,2,\ldots,m}\{d_i\}$ represents the maximum number of elements. Then the time complexity of calculating $P(S')$ is reduced from $O(mn)$ (calculating $P(S')$ from scratch) to $O(|d_{max}|)$.

Although the general union neighborhood $N_1 \cup N_2$ has been employed in (Li et al., 2021), the proposed IHS algorithm adopts a different way to explore $N_1 \cup N_2$ much more efficiently. First, IHS uses a filtering strategy to exclude some unpromising solutions directly. Second, the $gain\_updating$ strategy is adopted to accelerate the evaluation of each neighbor solution in the union neighborhood. Third, IHS employs the $aspiration\ criterion$ to accept conditionally a best neighbor solution forbidden by the tabu list (see Section 2.4.2), while such a rule was not applied in (Li et al., 2021).

2.4.2. *Tabu list management*

The TS procedure uses the tabu list T to avoid revisiting previously seen solutions. Usually, when a move operation ($flip$ or $swap$) is performed, the involved items are recorded in T and forbidden to be changed in the next several consecutive iterations. In our case, the reverse move is forbidden throughout the current round of tabu search. Particularly, for the $swap$ operator, we use a $m \times m$ matrix to record the paired items i and j. A performed $swap(i,j)$ or $swap(j,i)$ move is forbidden to take part in a move again. For the $flip$ move involving only one item i, the item will be prohibited to move back to its original (selected or non-selected) set in the current round of the TS procedure. Thus, item i is not allowed to take part in a $flip$ move again, but remains eligible for the $swap$ operation. Our preliminary experiment indicates that this tabu list management is more effective than the previous technique applied in (Li et al., 2021). Moreover, we also employ the $aspiration\ criterion$ (Glover and Laguna, 1997) to allow a forbidden move if the move leads to a solution better than the best solution S_b.

2.5. Hyperplane search

From the best feasible solution $S_b = (x_1, \ldots, x_m)$ from the TS procedure, we obtain the starting hyperplane dimension $k = \sum_{i=1}^{m} x_i$ (i.e., the number of selected items in S) (line 7, Alg. 1). Then as shown in Algorithm 4), several hyperplanes $k, k+1, \ldots, k+\gamma$ (γ is the limit of hyperplane search) are successively explored (lines 8-30, Alg. 4). For a given hyperplane, the hyperplane search procedure examines both feasible and infeasible solutions in the associated space. When the search on the current hyperplane terminates, the last solution is extended with a new random item (line 29, Alg. 4), which is used to seed the search on the next hyperplane $k+1$. This process stops when the last hyperplane $k+\gamma$ is examined.

For a given hyperplane dimension, the hyperplane search procedure performs the inner "while" loop (lines 8-27, Alg. 4) to examine candidate solutions. For this, it explores the following unconstrained $swap$ neighborhood, which includes both feasible and infeasible solutions.

$$N_3(S) = \{S' : S' = S \oplus swap(q,p), q \in V, p \in \bar{V}, 1 \leq i \leq m\} \quad (11)$$

where V and $\bar{V}$ represent selected and non-selected items in S, respectively.

To perform a transition from S to a neighbor solution S' in $N_3(S)$, the hyperplane search procedure follows the following steps (lines 9-19, Alg. 4). First, the minimum overweight value W_{omin} and the maximum objective value f_{max} are initialized at first (line 12, Alg. 4). Then, a subset $N'(S) \subseteq N(S)$ of neighbor solutions are identified satisfying one of two conditions (lines 13-15, Alg. 4): (1) the overweight value $W_{S'}$ is smaller than the minimum overweight W_{omin}; (2) $W_{S'}$ is equal to W_{omin}, while $f(S')$ is better than the maximum objective value f_{max} found so far. Considering that there may be several neighbor solutions with an equal $W_{S'}$ value and $f(S')$ value in the neighborhood, we use the set $N'(S)$ to save these equal neighbor solutions. Finally, the best neighbor solution S' with the largest $f(S')$ value or the least $W_o(S')$ value in $N'(S)$ is selected (ties breaking randomly) to replace the current solution S (line 19, Alg. 4). As a result, the total weight $W(S)$ of S always remains near the given budget, which favors the finding of high-quality feasible solutions within the feasible region and not-too-bad infeasible solutions close to the feasibility boundary. After each solution transition, the best solution s_b encountered during hyperplane search and the counter i are updated if needed (lines 20-24, Alg. 4).

Note that our hyperplane search does not require an additional penalty function for evaluating infeasible solutions, making our approach simpler compared to the conventional penalty function approach such as (Hamida and Schoenauer, 2000; Sun, Hao, Lai and Wu, 2018).

To prevent search cycling, the hyperplane search procedure uses the same tabu list technique as presented in Section 2.4.2.

Finally, during the hyperplane search, only feasible neighbor solution S' better than S_b that satisfies the budget constraint is accepted to update the best feasible solution found S_b (lines 21-22, Alg. 4). As a result, the IHS algorithm always returns a feasible solution S_b at the end of the hyperplane search procedure.

2.6. Greedy randomized perturbation procedure

After the hyperplane search phase, the IHS algorithm diversifies its search process by performing a greedy randomized perturbation. Basically, this perturbation procedure modifies the last solution S from hyperplane search by dropping some items and then adding some other items. The selection of the dropped and added items is based on the *contribution* CT and the associated dynamic density DD of each item (see Section 2).

As shown in Algorithm 5, the perturbation procedure includes two "while" loops. The first "while" loop (lines 4-10, Alg. 5) is devoted to remove some unpromising items from the solution. For this, we calculate for each selected item u in S, its *contribution* value CT, dynamic density value DD_u and probability P_u. Then, we iteratively drop items u from S according to the probability P_u until $|S| \times \delta_{max}$ items are removed, where δ_{max} is a parameter. The second "while" loop (lines 11-18, Alg. 5) adds non-selected items randomly until the given budget is met.

Algorithm 4 Hyperplane search procedure

```
1: Input: Input solution $S$, starting hyperplane $k$, limit of hyperplane search $\gamma$, search depth for one hyperplane $I_{max2}$, neighborhood $N_3$.
2: Output: The best feasible solution $S_b$ found during the hyperplane search and the last local optimum $S_l$ found by the hyperplane search.
3: $S_b \leftarrow S$
4: $K \leftarrow k + \gamma$
   /* The hyperplane search explores hyperplanes $k, k+1, \dots, k+\gamma$ */
5: while $k \leq K$ do
6:     $i \leftarrow 0$
7:     Initialize $tabu\_list$
8:     while $iter \leq I_{max2}$ do
9:         $(W_{omin}, f_{max}) \leftarrow (\infty, -\infty)$
10:        $N'(S) \leftarrow \emptyset$
11:        for Each admissible neighbor solution $S'$ in $N_3(S)$ do
12:            Calculate the overweight value $W_o(S')$ by Equation 7
13:            if $(W_o(S') < W_{omin}) \vee (W_o(S') = W_{omin} \wedge f(S') > f_{max})$ then
14:                $(W_{omin}, f_{max}) \leftarrow (W_o(S'), f(S'))$
15:                $N'(S) \leftarrow S'$
16:            end if
17:        end for
18:        Select the best neighbor solution $S' \in N'(S)$ with the least $W_o(S')$ or with the largest $f(S')$
19:        $S \leftarrow S'$
20:        if $(f(S) > f(S_b)) \wedge (W(S) \leq C)$ then
21:            $S_b \leftarrow S$
22:            $i \leftarrow 0$
23:        else
24:            $i \leftarrow i + 1$
25:        end if
26:        Update $tabu\_list$
27:    end while
28:    $k \leftarrow k + 1$                    /* Move to the next hyperplane */
29:    $S \leftarrow Random\_add\_one\_item(S)$ /* Create the starting solution of next hyperplane search */
30: end while
31: return $S_b$
```

Algorithm 5 Greedy randomized perturbation procedure

```
1: Input: Input solution $S$.
2: Output: The perturbed solution $S$, perturb strength $\delta_{max}$.
3: $\delta \leftarrow 0$
4: while $\delta < |S| \times \delta_{max}$ do
5:     $CT \leftarrow Calculate\_contribution(S)$
6:     $DD \leftarrow Calculate\_dynamic\_density(CT)$
7:     $P \leftarrow Calculate\_probability(DD)$
8:     $S \leftarrow Drop\_one\_item(P, S)$
9:     $\delta \leftarrow \delta + 1$
10: end while
11: while $W(S) \leq C$ do
12:     Random choose one non-selected item $i$
13:     if $w(i) + W(S) \leq C$ then
14:         $S \leftarrow Add\_item(i, S)$
15:     else
16:         break;
17:     end if
18: end while
19: return $S$
```

2.7. Complexity analysis

For the dynamic greedy initialization procedure, the main "while" loop (lines 5-14, Alg. 2) can be achieved in

$O(m^2n)$. Given the maximum iterations I_i of the initialization procedure, the corresponding time complexity is $O(m^2n \times I_i)$. Given the selected and non-selected item sets V and $\bar{V}$, the main iteration of the TS procedure can be achieved in $O((m + |V| \times |\bar{V}|) \times n)$. Then the time complexity of the TS procedure is $O((m + |V| \times |\bar{V}|) \times n \times I_{max1})$, where I_{max1} is its maximum iterations. Since only the *swap* operator is adopted in the hyperplane search procedure, the corresponding time complexity can be expressed by $O(|V| \times |\bar{V}| \times n \times I_{max2})$, where I_{max2} is the search depth for one hyperplane. The greedy randomized perturbation procedure can be achieved in $O(m^2n \times |V|)$. Let I_{max} be the maximum iterations of IHS, the overall time complexity of our algorithm is $O(m^2n \times (I_{max1} + I_{max2}) \times I_{max})$.

3. Results and comparisons

We now evaluate the IHS algorithm by testing it on benchmark instances and comparing its results with the literature.

3.1. Benchmark instances and experimental settings

Benchmark. The 30 benchmark instances[1] were proposed in (Li et al., 2021) with a range of 585 to 1000 items and elements. Given a relationship matrix, where $M_{ij} = 1$ indicates that the item i covers the element j. Let $\alpha = (\sum_{i=1}^{m} \sum_{j=1}^{n} M_{ij})/(mn)$ be the density of $M_{ij} = 1$ in the relationship matrix and C be the given budget. Then the BMCP instance can be denoted by $m\_n\_\alpha\_C$. For the 30 instances, α is set to 0.05 or 0.075 and C is set to 1500 or 2000.

Computing platform. Our algorithm is written in C++ and compiled using the g++ compiler with the -O3 option. The experiments are performed on an Intel Xeon E5-2670 computer with 2.5 GHz CPU and 2 GB RAM running Linux.

Parameter settings. IHS requires four parameters: depth of tabu search I_{max1}, depth of hyperplane search I_{max2}, limit of hyperplane search γ, perturbation strength δ. Intuitively, the value of γ should not be too large due to the budget constraint. In our case, γ is set to 2 and we analyze the influence of γ in Section 4.1. For the other three parameters, we employ the "Irace" tool (López-Ibáñez, Dubois-Lacoste, Cáceres, Birattari and Stützle, 2016) to automatically determine their values. Our tuning experiment is based on 7 representative instances (See Section 4) with a cut-off time of 200s. Table 1 shows the candidate values and the final values of the parameters recommended by "Irace".

Reference algorithms. For the comparative studies, we employ the best approximate algorithm (AA) (Khuller et al., 1999) and the most recent probability learning based tabu search algorithm (PLTS) (Li et al., 2021). As shown in (Li et al., 2021), PLTS is the current best heuristic algorithm for BMCP. We additionally include the best lower bounds (LB) and upper bounds (UB) from (Li et al., 2021) from the CPLEX solver under the running time of 5 hours. Note that IHS and PLTS use the same computing platform, which ensures a fair comparison.

Table 1
Parameter settings of IHS.

Para.	Sect.	Description	Considered values	Value
I_{max1}	2.4	depth of tabu search	$\{1000, 1500, 2000, 2500, 3000)\}$	2000
I_{max2}	2.5	depth of hyperplane search	$\{100, 150, 200, 250, 300\}$	250
δ	2.6	perturbation strength	$\{0.3, 0.4, 0.5, 0.6, 0.7\}$	0.6
γ	2.5	limit of hyperplane search	-	2

Stopping condition. The cut-off time of the IHS algorithm is set to 600s, which is consistent with the reference algorithms. Considering the stochastic nature of the algorithm, each instance is independently solved 30 times by IHS with different random seeds, like (Li et al., 2021).

3.2. Computational results

Table 2 summarizes our results and shows a comparison with the reference algorithms on the 30 BMCP benchmark instances[2]. Columns 1 presents the instance name of BMCP. The following two columns provide the lower bounds (LB) and upper bounds (UB) obtained by CPLEX. The remaining columns provide the detailed results of the compared algorithms, based on four performance indicators: the best objective value (f_{best}), the average objective value (f_{avg}), the standard deviations (std) and the average run time t_{avg} (seconds) to reach f_{best}. To highlight the very best results, the dominating values for the performance indicators f_{best} and f_{avg} are indicated in **bold** and the equal results are marked in *italic*. Moreover, the last row presents the average value of each column.

Table 2 indicates that IHS surpasses the reference algorithms according to all the performance indicators. IHS discovers 18 new lower bounds (f_{best} values) and obtains equal f_{best} results for the 12 remaining instances. Compared to the AA algorithm, IHS obtains better f_{best} values for the 30 instances without exception. When compared with the main reference algorithm PLTS, our IHS algorithm is still highly competitive, according to both the f_{best} and the f_{avg} values. Moreover, we observe that IHS is highly robust since it can achieve 100% success rate for 26 instances (with $std = 0$). Furthermore, The small values of average run time t_{avg} in the last row reveal that our algorithm is computationally efficient.

To better illustrate the results of the compared algorithms, we report the summarized comparisons of IHS against each reference algorithm in Table 3. This table provides the number of instances for which IHS achieves a better, equal or worse result compared to each reference algorithm (corresponding to columns 3 to 5). We also

[1]The 30 benchmark instances are available at: `https://github.com/Zequn-Wei/BMCP`.

[2]The solution certificates are available at the GitHub link of footnote 1. The code of IHS will also be available upon the publication of the paper.

Table 2
Computational results and comparisons of the IHS algorithm with the reference algorithms.

Instance	CPLEX (5 hours)		AA (Khuller et al., 1999)	PLTS (Li et al., 2021)				IHS (this work)			
	LB	UB	f_{best}	f_{best}	f_{avg}	Std	t_{avg}	f_{best}	f_{avg}	std	t_{avg}
585_600_0.05_2000	70742	74224.94	70494	*71102*	71065.17	82.36	309.602	*71102*	**71097.23**	5.45	13.454
585_600_0.075_1500	69172	76716.90	68475	70677	70677.00	0.00	61.242	**71025**	**71025**	0	28.006
600_600_0.05_2000	68477	72880.93	66095	*68738*	68472.00	71.09	95.638	*68738*	**68738**	0	0.162
600_600_0.075_1500	71018	76337.67	70445	71746	71746.00	0.00	27.975	**71904**	**71904**	0	17.295
600_585_0.05_2000	66452	71094.27	67256	*67636*	67460.80	350.40	202.660	*67636*	**67636**	0	0.041
600_585_0.075_1500	70113	76332.85	68005	*70588*	70406.63	105.09	584.205	*70588*	**70588**	0	82.744
685_700_0.05_2000	80783	88447.67	79778	*81227*	80585.73	508.37	522.060	*81227*	**81227**	0	47.099
685_700_0.075_1500	81639	91378.98	80457	82955	82951.40	19.39	109.670	**83286**	**83175.67**	156.03	234.167
700_700_0.05_2000	77056	84855.44	76552	78028	77859.27	75.16	127.445	**78458**	**78458**	0	4.958
700_700_0.075_1500	81645	92151.99	83400	*84576*	84375.70	550.91	196.995	*84576*	**84576**	0	0.029
700_685_0.05_2000	77176	82815.11	76600	*78054*	78037.00	51.00	197.590	*78054*	**78054**	0	7.403
700_685_0.075_1500	76033	86566.79	75224	*78869*	*78869*	0	46.987	*78869*	*78869*	0	5.199
785_800_0.05_2000	91319	101585.69	90975	92608	92587.60	34.30	252.589	**92740**	**92740**	0	175.372
785_800_0.075_1500	92358	106842.67	90786	94245	94245.00	0.00	248.128	**95221**	**95221**	0	0.017
800_800_0.05_2000	89872	100373.77	89582	*91795*	91576.27	309.05	307.274	*91795*	**91795**	0	5.292
800_800_0.075_1500	94049	108005.62	93115	95533	95509.60	70.20	239.146	**95995**	**95995**	0	33.042
800_785_0.05_2000	86813	97477.34	86750	*89138*	88581.20	103.40	204.141	*89138*	**89138**	0	0.116
800_785_0.075_1500	89229	102867.98	90548	91021	91010.20	25.67	297.211	**91856**	**91856**	0	0.314
885_900_0.05_2000	99845	113746.97	99498	102162	101331.53	174.95	206.025	**102277**	**102277**	0	4.427
885_900_0.075_1500	102933	122093.51	105793	106577	105942.43	334.18	489.396	**106940**	**106940**	0	4.378
900_900_0.05_2000	100412	114551.09	98893	101265	101231.17	62.94	325.683	**102055**	**101727.73**	277.52	348.347
900_900_0.075_1500	101035	119626.49	103795	104521	104521.00	0	176.865	**105081**	**105081**	0	0.623
900_885_0.05_2000	96945	109948.52	98337	98840	98718.00	151.34	227.976	**99590**	**99590**	0	0.359
900_885_0.075_1500	99888	118554.77	100359	*105141*	104397.93	691.61	229.644	*105141*	**105141**	0	55.619
985_1000_0.05_2000	107488	124487.37	108105	109567	109408.77	227.85	212.193	**110669**	**110669**	0	29.281
985_1000_0.075_1500	111177	133789.78	113137	114969	113838.07	509.34	485.677	**115505**	**115505**	0	27.238
1000_1000_0.05_2000	111155	128583.63	111786	112802	111897.07	636.78	577.668	**113331**	**113316.20**	29.60	61.536
1000_1000_0.075_1500	115824	137900.40	118869	*120246*	118467.87	546.67	279.220	*120246*	**120246**	0	0.307
1000_985_0.05_2000	110134	125574.84	107548	111859	111228.80	828.72	244.920	**112057**	**112057**	0	155.882
1000_985_0.075_1500	108801	130981.38	111778	112250	112125.87	143.22	234.614	**113615**	**113615**	0	0.251
#Avg	89986.10	102359.84	90081.17	91957.83	91637.47	222.13	257.348	**92290.50**	**92275.26**	15.62	44.765

Table 3
Summarized comparisons of IHS against each reference algorithm.

Algorithm pair	Indicator	#Wins	#Ties	#Losses	*p-value*
IHS vs. AA (Khuller et al., 1999)	f_{best}	30	0	0	1.73e-06
IHS vs. PLTS (Li et al., 2021)	f_{best}	18	12	0	1.96e-04
	f_{avg}	29	1	0	2.56e-06

perform the Wilcoxon signed-rank test to assess the statistical difference between the compared algorithms. From Table 3, we observe that IHS fully dominates AA. IHS also performs better than PLTS, by reporting 18 better f_{best} values and 29 better f_{avg} values. Moreover, the small *p-values* ($\ll$ 0.05) in the last column confirm the differences of IHS against the reference algorithms are statistically significant.

Furthermore, Figure 1 completes the comparison with the performance profiles (see (Dolan and Moré, 2002) for more details) of the compared algorithms according to the f_{best} values (top) and the f_{avg} values (bottom). Given the set $\mathcal{P}$ of instances tested and the set $\mathcal{A}$ of compared algorithms, the performance ratio can be denoted by $r_{p,a} = \frac{f_{p,a}}{max\{f_{p,a}:a\in\mathcal{A}\}}$, where $f_{p,a}$ is the objective value (f_{best} or f_{avg}) of instance p attained by algorithm a. In Figure 1, the intersection of each curve with the Y-*axis* indicates the fraction of the corresponding algorithm a can achieve the best results among all the algorithms in $\mathcal{A}$. From Figure 1, we can clearly observe that the curve of IHS lies strictly above the curves of AA and PLTS. This indicates that

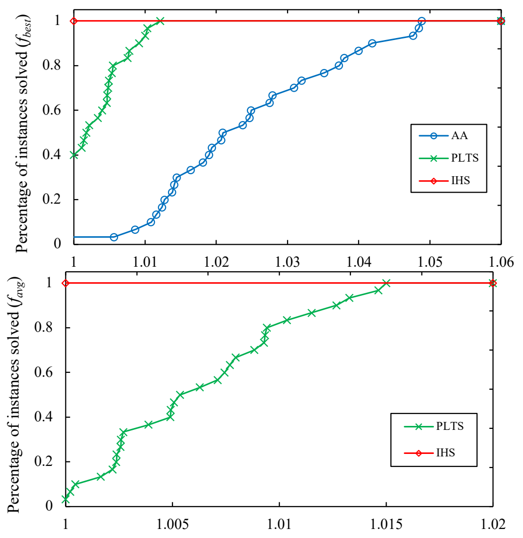

Figure 1: Performance profiles of the compared algorithms according to the f_{best} values (top) and the f_{avg} values (bottom).

compared to AA and PLTS, the IHS algorithm obtains a greater cumulative probability of attaining the best results for the tested instances. This comparison confirms that IHS is highly effective and robust compared to AA and PLTS.

4. Analysis and discussions

This section provides several experiments to study the key components of the IHS algorithm.

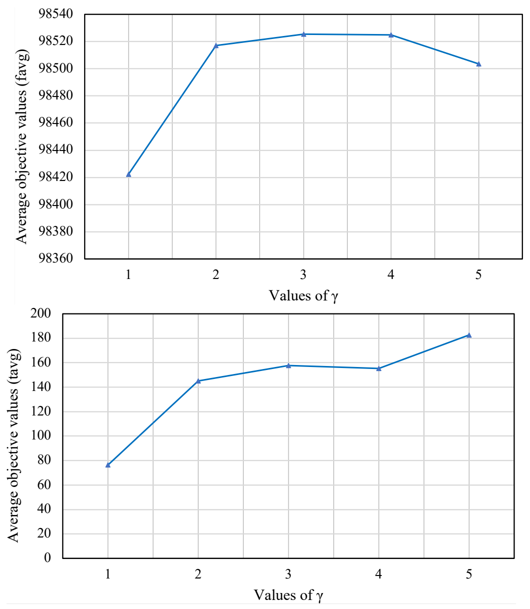

Figure 2: Performances of the IHS algorithm with different γ values in terms of the f_{avg} values (top) and the t_{avg} values (bottom).

4.1. Analysis of parameters

We perform a 2-level full factorial experiment (Montgomery, 2017) to study the interaction effects of the main parameters of IHS: depth of tabu search I_{max1} (Section 2.4), depth of hyperplane search I_{max2} (Section 2.5), and perturbation strength δ (Section 2.6). This study is based on seven selected instances of different sizes and features: 585_600_0.05_2000, 685_700_0.075_1500, 785_800_0.05_2000, 900_900_ 0.05_2000, 985_1000_0.075_1500, 1000_985_0.05_2000, 1000_1000_0.05_2000.

Specifically, the boundary values of each parameter tested in Table 1 are adopted as the high level and low level of this experiment. Thus, we obtain 8 ($2^3 = 8$) combinations of the parameters. For each instance, we run IHS with each combination 30 times. Then the average results of f_{best} on the seven instances are used for the analysis. The *p-value* of 0.695 from the analysis of variance discloses that there are no statistically significant interaction effects among the three parameters.

Now we investigate the impacts of the parameter γ. Due to the budget constraint of BMCP, a too large value of γ will always guide IHS to infeasible spaces, which is obviously unsuitable. For this experiment, γ ranges from 1 to 5 with a step size of 1. We run IHS with these five γ values to solve the seven selected instances with the default settings of Section 3.1. The results shown in Figure 2 indicate that IHS reports similar average result f_{avg} and average run time t_{avg} with $\gamma = 2, 3, 4$ and obtains worse results with $\gamma = 1, 5$. Without surprise, when $\gamma = 5$, IHS takes a longer time to get worse results. This experiment justifies the choice of $\gamma = 2$ as the default setting of this parameter.

4.2. Effectiveness of the hyperplane search

We now assess the effectiveness of the hyperplane search strategy, which is an important component of the proposed algorithm. Specifically, we construct an IHS variant (denoted by IHS$^-$) by disabling the hyperplane search component by setting γ to 0. As a result, IHS$^-$ will only explore the feasible and infeasible regions on the current hyperplane k identified by the tabu search procedure (see Alg. 1). This experiment is carried out by running IHS and IHS$^-$ to solve the 30 instances according to default experimental settings. Table 4 shows the results of this experiment including the f_{best}, f_{avg} and std values. In the last two rows, we provide the average value of each column (#Avg) and the number of instances for which each variant reaches the best results in each column (#Best).

Table 4 shows that IHS performs better with the hyperplane search procedure. Specifically, IHS achieves better f_{best} results for 8 instances (highlighted in bold) and equal f_{best} values for the remaining instances. In terms of f_{avg}, IHS performs better than IHS$^-$ on 14 instances. IHS has also better standard deviations, implying that it is more robust than IHS$^-$. Finally, the *p-value* smaller than 0.05 discloses that there is a significant difference between IHS and IHS$^-$. These outcomes demonstrate the benefit of the hyperplane search strategy of the IHS algorithm.

4.3. Convergence analysis

To investigate the running behavior of IHS with the main reference algorithm PLTS (Li et al., 2021), we conduct an additional convergence analysis. This experiment is carried out on two difficult instances 685_700_0.075_1500 and 900_900_0.05_2000. We run each algorithm 30 times to solve each instance under 600 seconds per run. The convergence graphs (i.e., the running profiles) shown in Figure 3 are obtained by the function: $t \rightarrow f$ where t is the running time and f is the best objective value achieved by each algorithm at time t.

Figure 3 confirms the dominance of IHS over PLTS since the curve of IHS strictly runs above the curve of PLTS, implying that not only IHS converges faster than PLTS, but also converges better. These outcomes confirm the competitiveness of IHS compared to the leading reference algorithm PLTS.

4.4. Time-to-target analysis

We carry out a time-to-target analysis (TTT) (Aiex, Resende and Ribeiro, 2007; Ribeiro, Rosseti and Campos, 2012) to further compare the computational efficiency of IHS and the main reference algorithm PLTS. Specifically, we analyze the required time for IHS and PLTS to achieve a given target objective value. This experiment is based on two benchmark instances of large size (see in Figure 4). The target value of these two instances is set to 111000 and 109400, respectively. For the experiment, we run IHS and PLTS 100 times independently with the default experimental settings. Each run is terminated immediately when the given target value is reached. The corresponding

Table 4
Comparison of the IHS algorithm with (IHS) and without (IHS⁻) the hyperplane search strategy.

Instance	IHS			IHS⁻		
	f_{best}	f_{avg}	*Std*	f_{best}	f_{avg}	*std*
585_600_0.05_2000	*71102*	*71097.23*	5.45	*71102*	*71097.23*	5.45
585_600_0.075_1500	**71025**	**71025**	0	70677	70654.80	59.80
600_600_0.05_2000	*68738*	*68738*	0	*68738*	*68738*	0
600_600_0.075_1500	**71904**	**71904**	0	71746	71746	0
600_585_0.05_2000	*67636*	*67636*	0	*67636*	*67636*	0
600_585_0.075_1500	**70588**	**70588**	0	70318	70318	0
685_700_0.05_2000	*81227*	**81227**	0	*81227*	81084.20	218.13
685_700_0.075_1500	**83286**	**83175.67**	156.03	82955	82955	0
700_700_0.05_2000	*78458*	*78458*	0	*78458*	*78458*	0
700_700_0.075_1500	*84576*	*84576*	0	*84576*	*84576*	0
700_685_0.05_2000	*78054*	**78054**	0	*78054*	78043.20	58.16
700_685_0.075_1500	*78869*	*78869*	0	*78869*	*78869*	0
785_800_0.05_2000	**92740**	**92740**	0	92608	92531.27	133.10
785_800_0.075_1500	*95221*	*95221*	0	*95221*	*95221*	0
800_800_0.05_2000	*91795*	*91795*	0	*91795*	*91795*	0
800_800_0.075_1500	**95995**	**95995**	0	95533	95533	0
800_785_0.05_2000	*89138*	*89138*	0	*89138*	*89138*	0
800_785_0.075_1500	*91856*	*91856*	0	*91856*	*91856*	0
885_900_0.05_2000	*102277*	*102277*	0	*102277*	*102277*	0
885_900_0.075_1500	*106940*	*106940*	0	*106940*	*106940*	0
900_900_0.05_2000	**102055**	**101727.73**	277.52	101265	101265	0
900_900_0.075_1500	*105081*	*105081*	0	*105081*	*105081*	0
900_885_0.05_2000	*99590*	*99590*	0	*99590*	*99590*	0
900_885_0.075_1500	*105141*	**105141**	0	*105141*	103302.30	612.90
985_1000_0.05_2000	*110669*	**110669**	0	*110669*	110598.30	212.42
985_1000_0.075_1500	*115505*	**115505**	0	*115505*	115436.47	174.73
1000_1000_0.05_2000	*113331*	**113316.20**	29.60	*113331*	113308.80	33.91
1000_1000_0.075_1500	*120246*	*120246*	0	*120246*	*120246*	0
1000_985_0.05_2000	**112057**	**112057**	0	111859	111847.33	21.15
1000_985_0.075_1500	*113615*	*113615*	0	*113615*	*113615*	0
#Avg	**92290.50**	**92275.26**	15.62	92200.87	92125.23	50.99
#Best	**30**	**30**	-	22	16	-

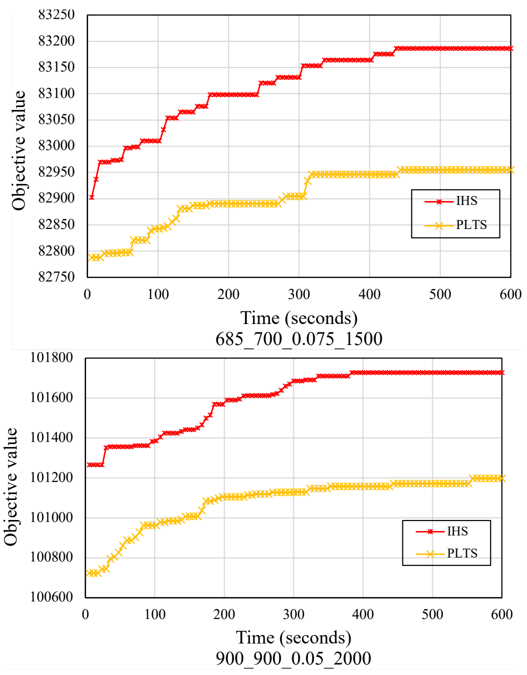

Figure 3: Convergence analysis of IHS and the reference algorithm PLTS (Li et al., 2021).

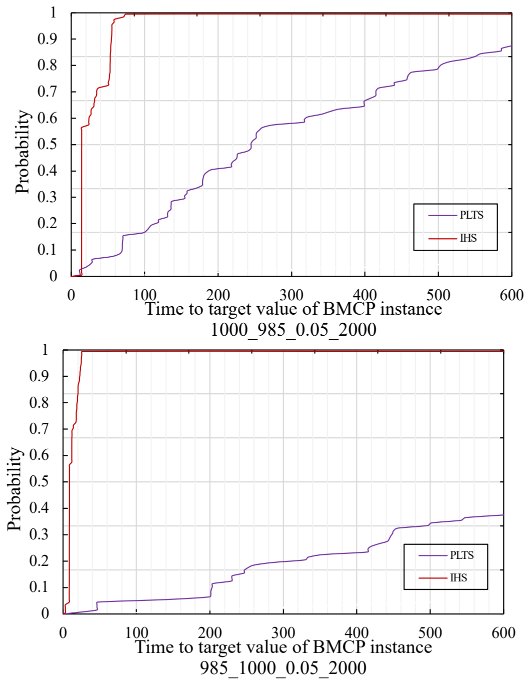

Figure 4: Time-to-target analysis of IHS and the reference algorithm PLTS (Li et al., 2021).

running time is served as the output of the experiment. The results of this TTT analysis are provided in Figure 4. The X and Y axes shows the running time and the cumulative probability to reach the given target, respectively.

As shown in Figure 4, the TTT lines of IHS lie almost strictly above the lines of PLTS, indicating that IHS is able to reach the given target with a significantly higher cumulative probability. This experiment again indicates the superiority of the IHS algorithm in terms of computational efficiency.

5. Conclusions and perspectives

The budgeted maximum coverage problem is a relevant model for various applications. We introduced the first iterated hyperplane search algorithm, which combines effective tabu search within the feasible space, mixed hyperplane search examining both feasible and infeasible regions and perturbation-driven diversification.

The algorithm is assessed on 30 benchmark instances commonly tested previously. Computational results disclose that our algorithm is superior to the state-of-the-art algorithms. Especially, the algorithm obtains new record results (improved lower bounds) for 18 instances. Furthermore, considering that BMCP is a relevant model to formulate several practical problems, the proposed algorithm and its code can help to better solve these applications.

For future work, it would be meaningful to study other techniques for identifying promising hyperplanes. Also, since each hyperplane defines a subproblem of BMCP, it would be interesting to check whether better results can be obtained when an exact algorithm or a general mixed integer programming solver is adopted in the hyperplane search procedure. More generally, existing studies focused on practical heuristic algorithms, whose solution quality cannot be theoretically guaranteed. To obtain solutions of guaranteed quality, exact and approximation algorithms are needed. However, dedicated exact algorithms for BMCP, which can theoretically provide optimal solutions, are still missing and there is only one approximation algorithm with an approximation ratio of $(1 - 1/e)$ (Khuller et al., 1999). Therefore, research on these approaches is needed to design practically applicable exact and approximation algorithms, which can hopefully provide optimal solutions or tight bounds for the benchmark instances.

Acknowledgments

We are grateful to the reviewers for their valuable comments and suggestions. This work was partially supported by the Fundamental Research Funds for the Central Universities (Grant No. 2022RC64). Support from the High-performance Computing Platform of BUPT is also acknowledged.

References

Aïder, M., Gacem, O., Hifi, M., 2022. A hybrid population-based algorithm for the bi-objective quadratic multiple knapsack problem. Expert Systems with Applications 191, 116238.

Aiex, R.M., Resende, M.G.C., Ribeiro, C.C., 2007. TTT plots: a perl program to create time-to-target plots. Optim. Lett. 1, 355–366.

Avci, M., Topaloglu, S., 2017. A multi-start iterated local search algorithm for the generalized quadratic multiple knapsack problem. Computers & Operations Research 83, 54–65.

Cachon, G., Terwiesch, C., 2011. Matching supply with demand: An introduction to operations management. McGraw-Hill.

Chauhan, D., Unnikrishnan, A., Figliozzi, M., 2019. Maximum coverage capacitated facility location problem with range constrained drones. Transportation Research Part C: Emerging Technologies 99, 1–18.

Chekuri, C., Kumar, A., 2004. Maximum coverage problem with group budget constraints and applications, in: Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques. Springer, pp. 72–83.

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

Cohen, R., Katzir, L., 2008. The generalized maximum coverage problem. Information Processing Letters 108, 15–22.

Curtis, D.E., Pemmaraju, S.V., Polgreen, P., 2010. Budgeted maximum coverage with overlapping costs: monitoring the emerging infections network, in: Proceedings of the 12th Workshop on Algorithm Engineering and Experiments, SIAM. pp. 112–123.

Dolan, E.D., Moré, J.J., 2002. Benchmarking optimization software with performance profiles. Mathematical Programming 91, 201–213.

Fleszar, K., Hindi, K.S., 2009. Fast, effective heuristics for the 0–1 multi-dimensional knapsack problem. Computers & Operations Research 36, 1602–1607.

Glover, F., Laguna, M., 1997. Tabu search. Springer Science+Business Media New York.

Goldschmidt, O., Nehme, D., Yu, G., 1994. Note: On the set-union knapsack problem. Naval Research Logistics 41, 833–842.

Hamida, S.B., Schoenauer, M., 2000. An adaptive algorithm for constrained optimization problems, in: Proceedings of the 6th International Conference Parallel Problem Solving from Nature, Lecture Notes in Computer Science 1917, Springer. pp. 529–538.

He, Y., Xie, H., Wong, T.L., Wang, X., 2018. A novel binary artificial bee colony algorithm for the set-union knapsack problem. Future Generation Computer Systems 78, 77–86.

Hochba, D.S., 1997. Approximation algorithms for NP-hard problems. ACM Sigact News 28, 40–52.

Kar, B., Wu, E.H.K., Lin, Y.D., 2016. The budgeted maximum coverage problem in partially deployed software defined networks. IEEE Transactions on Network and Service Management 13, 394–406.

Kartashov, O., SergiyYakoIlev, O.S., YakoIleva, I., Shekhovtsov, S., 2019. Modeling and simulation of coverage problem in geometric design systems, in: Proceedings of the IEEE 15th International Conference on the Experience of Designing and Application of CAD Systems, pp. 20–23.

Khuller, S., Moss, A., Naor, J.S., 1999. The budgeted maximum coverage problem. Information Processing Letters 70, 39–45.

Li, L., Wang, D., Li, T., Knox, D., Padmanabhan, B., 2011. Scene: a scalable two-stage personalized news recommendation system, in: Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 125–134.

Li, L., Wei, Z., Hao, J.K., He, K., 2021. Probability learning based tabu search for the budgeted maximum coverage problem. Expert Systems with Applications , 115310.

Liang, D., Shen, H., Chen, L., 2021. Maximum target coverage problem in mobile wireless sensor networks. Sensors 21, 184.

Lin, G., Guan, J., Li, Z., Feng, H., 2019. A hybrid binary particle swarm optimization with tabu search for the set-union knapsack problem. Expert Systems with Applications 135, 201–211.

Liu, X., He, Y., 2019. Estimation of distribution algorithm based on lévy flight for solving the set-union knapsack problem. IEEE Access 7, 132217–132227.

López-Ibáñez, M., Dubois-Lacoste, J., Cáceres, L.P., Birattari, M., Stützle, T., 2016. The irace package: Iterated racing for automatic algorithm configuration. Operations Research Perspectives 3, 43–58.

Montgomery, D.C., 2017. Design and analysis of experiments. John Wiley & Sons.

Ozsoydan, F.B., Baykasoglu, A., 2019. A swarm intelligence-based algorithm for the set-union knapsack problem. Future Generation Computer Systems 93, 560–569.

Ribeiro, C.C., Rosseti, I., Campos, R.V., 2012. Exploiting run time distributions to compare sequential and parallel stochastic local search algorithms. J. Global Optimization 54, 405–429.

Simchi-Levi, D., 2010. Operations rules: delivering customer value through flexible operations. MIT Press.

Suh, K., Guo, Y., Kurose, J., Towsley, D., 2006. Locating network monitors: complexity, heuristics, and coverage. Computer Communications 29, 1564–1577.

Sun, W., Hao, J.K., Lai, X., Wu, Q., 2018. Adaptive feasible and infeasible tabu search for weighted vertex coloring. Information Sciences 466, 203–219.

Takamura, H., Okumura, M., 2009. Text summarization model based on maximum coverage problem and its variant, in: Proceedings of the 12th Conference of the European Chapter of the ACL, pp. 781–789.

Taylor, R., 2016. Approximations of the densest k-subhypergraph and set union knapsack problems. arXiv preprint :1610.04935 .

Vasquez, M., Hao, J.K., 2001. A hybrid approach for the 0-1 multidimensional knapsack problem, in: Proceedings of the 17th International Joint Conference on Artificial Intelligence, Seattle, Washington, USA. pp. 328–333.

Wei, Z., Hao, J.K., 2020. Kernel based tabu search for the set-union knapsack problem. Expert Systems with Applications 165, 113802.

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