Iterated hyperplane search for the budgeted maximum coverage problem

Zequn Wei\textsuperscript{a}, Jin-Kao Hao\textsuperscript{b,}\textsuperscript{*}

\textsuperscript{a}School of Economics and Management, Beijing University of Posts and Telecommunications, 100876 Beijing, China
\textsuperscript{b}LERIA, Université d’Angers, 2 Boulevard Lavoisier, 49045 Angers, France


\textbf{ARTICLE INFO}

\textbf{Keywords:}
Budgeted maximum coverage
Knapsack
Heuristics
Iterated local search
Hyperplane search
Combinatorial optimization

\textbf{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, SergiyYakolliev, Yakollieva and Shekhovtsov, 2019; Liang, Shen and Chen, 2021). The budgeted maximum coverage problem (BMCP) generalizes the MCP. Given a set of $E = \{1, \ldots, n\}$ of $n$ elements and a set $I = \{1, \ldots, m\}$ 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 $I$ and the elements of $E$ are given by a $m \times n$ binary matrix $M$. Then BMCP aims to find a subset $S \subseteq I$ to maximize the profit sum $P(S)$ of the covered elements while ensuring that the weight sum $W(S)$ of the selected items satisfies a given budget (knapsack capacity) $C$ (Khuller, Moss and Naor, 1999). Formally, BMCP can be written as follows.

\begin{equation}
\text{Maximize} \quad P(S) = \sum_{j \in E_S} p_j
\end{equation}

\begin{equation}
\text{subject to} \quad W(S) = \sum_{i \in S} w_i \leq C
\end{equation}

\begin{equation}
S \subseteq I, \quad E_S = \bigcup_{i \in S} E_i
\end{equation}

with $S \subseteq I$ being the selected item set and $E_S \subseteq E$ being the elements covered by the item set $S$. 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 \textit{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...
We define the feasible search space $\Omega^F$ as the set of all feasible solutions satisfying the given budget $C$, i.e.,

$$\Omega^F = \{ x \in \{0, 1\}^m : \sum_{i=1}^m w_i x_i \leq C \} \quad (5)$$

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 $\Omega$ such that each solution satisfies the $k$-dimensional hyperplane constraint.

$$\Omega^{(k)} = \{ x \in \{0, 1\}^m : \sum_{i=1}^m x_i = k \} \quad (6)$$

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

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

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

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 $\Omega$. 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 $t_{\text{max}}$ 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 $l_{\text{max}}$ (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 $\Omega^{(k)}, \Omega^{(k+\gamma)}$ where $\gamma$ 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 $t_{\text{max}}$ is met while returning the recorded overall best solution $S^*$.

Algorithm 1 Iterated hyperplane search for BMCP

1: **Input:** Instance $I$, cut-off time $t_{\text{max}}$, depth of tabu search $I_{\text{max}}$, depth of hyperplane search $I_{\text{max}_2}$, 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: $S \leftarrow \text{Dynamic Greedy Initialization}(I)$ /* $S$ is the current solution */
4: $S^* \leftarrow S$
5: while $\text{Time} \leq t_{\text{max}}$ do
6: $S \leftarrow \text{Tabu Search}(S, N_1 \cup N_2, I_{\text{max}})$ /* Improve $S$ by tabu search */
7: $k \leftarrow \text{Determine Hyperplane Dimension}(S)$ /* Calculate the hyperplane dimension $k$ from the solution of tabu search */
8: $S \leftarrow \text{Hyperplane Search}(S, k, \gamma, N_3, I_{\text{max}_2})$ /* Improve $S$ by hyperplane search starting from hyperplane $k$ */
9: if $f(S) > f(S^*)$ then
10: $S^* \leftarrow S$
11: end if
12: $S \leftarrow \text{Greedy Randomized Perturbation}(S, \delta)$
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 $CT_u$ of the non-selected item $u$ to the solution $S$ by $CT_u = \sum_{j \in E_u \cap \bar{E}_S} \rho_j$ 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 $I$ 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.

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 $W(S) \leq C$ do
6: $CT \leftarrow \text{Calculate contribution()}$ /* Calculate the contribution of each non-selected item */
7: $DD \leftarrow \text{Calculate dynamic density}(CT)$ /* Calculate the dynamic density of each non-selected item */
8: Identify the item $u^*$ with the maximum density in $DD$
9: if $W(u^*) + W(S) \leq C$ then
10: $S \leftarrow \text{Add item}(u^*, S)$
11: else
12: break;
13: end if
14: end while
15: return $S$

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 $\Omega^F$. 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 $\Omega^F$ (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 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 $I_{\text{max}1}$ 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 $I_{\text{max}1}$.
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 $i \leq I_{\text{max}1}$ 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 $N'(S)$ */
10: $S' \leftarrow \text{argmax}(f(S'))$ /* The selected neighbor solution becomes the new current solution */
11: $S \leftarrow S'$
12: if $f(S) > f(S_b)$ then
13: $S_b \leftarrow f(S)$
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 $\Omega^F$, our TS procedures employs a constrained union neighborhood induced by two common move operators flip and swap (Li et al., 2021; Wei and Hao, 2021; Aider, Gacev 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 $S' = S \oplus \{\text{flip}, \text{swap}\}$ 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, respectively.

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

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

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 SUKIP (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 $d_{\text{max}} = \max_{i=1,2,\ldots,m}\{d_i\}$ 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 $O(d_{\text{max}})$.

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 $k, k+1, \ldots, k+\gamma$ ($\gamma$ 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.

$$N_3(S) = \{S' : S' = S \oplus \text{swap}(q,p), q \in V, p \in \bar{V}, 1 \leq i \leq m\} \quad (11)$$
where \( V \) and \( \tilde{V} \) represent selected and non-selected items in \( S \), respectively.

To perform a transition from \( S \) to a neighbor solution \( S' \) in \( N(S) \), the hyperplane search procedure follows the following steps (lines 9-19, Alg. 4). First, the minimum overweight value \( W_{\text{omin}} \) and the maximum objective value \( f_{\text{max}} \) 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 \( W_{\text{omin}} \); (2) \( W_S \) is equal to \( W_{\text{omin}} \), while \( f(S') \) is better than the maximum objective value \( f_{\text{max}} \) 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 \( W_{\text{omin}} \) value and \( f(S') \) value satisfying one of two conditions (lines 13-15, Alg. 4): (1) \( W_{\text{omin}} < W_{\text{omin}} \), \( f(S') > f_{\text{max}} \) then. Each admissible neighbor solution \( S' \) 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 \( S_b \) is 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 \( |S| \times \delta_{\text{max}} \) items are removed, where \( \delta_{\text{max}} \) 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 \( I_{\text{max}} \), neighborhood \( N \).
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: \( k \leftarrow k + \gamma \)
5: while \( k \leq K \) do
6: \( i \leftarrow 0 \)
7: Initialize tabu-list
8: while \( \text{iter} \leq I_{\text{max}} \) do
9: \( (W_{\text{min}}, f_{\text{max}}) \leftarrow (\infty, -\infty) \)
10: \( N'(S) \leftarrow \emptyset \)
11: for Each admissible neighbor solution \( S' \) in \( N(S) \) do
12: Calculate the overweight value \( W(S') \) by Equation 7
13: if \( W(S') < W_{\text{min}} \) then
14: \( W_{\text{min}} = W(S') \)
15: \( N'(S) \leftarrow S' \)
16: end if
17: end for
18: Select the best neighbor solution \( S' \in N'(S) \) with the least \( W(S') \) or with the largest \( f(S') \) \( S \leftarrow S' \)
19: if \( f(S) > f(S_b) \) then
20: \( S_b \leftarrow S \)
21: \( i \leftarrow 0 \)
22: else
23: \( i \leftarrow i + 1 \)
24: end if
25: Update tabu-list
26: end while
27: \( k \leftarrow k + 1 \)
28: \( f_{\text{max}} \leftarrow \max f(S') \)
29: \( S \leftarrow \text{Random_add_one_item}(S) \) /* Move to the next hyperplane */
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 \( \delta_{\text{max}} \).
3: \( \delta \leftarrow 0 \)
4: while \( \delta < |S| \times \delta_{\text{max}} \) do
5: \( CT \leftarrow \text{Calculate_contribution}(S) \)
6: \( DD \leftarrow \text{Calculate_dynamic_density}(CT) \)
7: \( P \leftarrow \text{Calculate_probability}(DD) \)
8: \( S \leftarrow \text{Drop_one_item}(P, S) \)
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: \( S \leftarrow \text{Add_item}(i, S) \)
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
$O(m^2 n)$. Given the maximum iterations $I$, of the initialization procedure, the corresponding time complexity is $O(m^2 n \times I)$. 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 $O(m^2 n \times |V|)$. Let $I_{max}$ be the maximum iterations of IHS, the overall time complexity of our algorithm is $O(m^2 n \times (I_{max1} + I_{max2}) \times I_{max})$.

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 \(\alpha = \frac{\sum_{m=1}^{n} \sum_{j=1}^{n} M_{ij}}{(mn)}\) 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 \(m_\text{\_n\_a\_C}\). For the 30 instances, \(\alpha\) 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 \(I_{\text{max1}}\), depth of hyperplane search \(I_{\text{max2}}\), limit of hyperplane search \(\gamma\), perturbation strength \(\delta\). Intuitively, the value of \(\gamma\) should not be too large due to the budget constraint. In our case, \(\gamma\) is set to 2 and we analyze the influence of \(\gamma\) in Section 4.1. For the other three parameters, we employ the “trace” tool (López-Iñáñez, 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 “trace”.

**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.

**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).

### Table 1

<table>
<thead>
<tr>
<th>Parameter settings of IHS.</th>
<th>Considered values</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$I_{\text{tabu}}$</td>
<td>Depth of tabu search</td>
<td>((100; 150; 200; 250; 300))</td>
</tr>
<tr>
<td>$I_{\text{hyp}}$</td>
<td>Depth of hyperplane search</td>
<td>((100; 150; 200; 250; 300))</td>
</tr>
<tr>
<td>$\delta$</td>
<td>Perturbation strength</td>
<td>((0.3, 0.4, 0.5, 0.6, 0.7))</td>
</tr>
<tr>
<td>$\gamma$</td>
<td>Limit of hyperplane search</td>
<td>-</td>
</tr>
</tbody>
</table>

\(^1\)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.
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 PLTS, by reporting 18 better \( f_{\text{best}} \) values and 29 better \( f_{\text{avg}} \) values. Moreover, the small \( p \)-values (\( p < 0.05 \)) 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 \( f_{\text{best}} \) values (left) and the \( f_{\text{avg}} \) values (right). Given the set \( P \) of instances tested and the set \( A \) of compared algorithms, the performance ratio can be denoted by

\[
r_{p,a} = \frac{f_{p,a}}{\max_{f_{p,a} \in A} f_{p,a}},
\]

where \( f_{p,a} \) is the objective value (\( f_{\text{best}} \) or \( f_{\text{avg}} \)) of instance \( p \) attained by algorithm \( a \). In Figure 1, the intersection of each curve with the \( Y \)-axis indicates the fraction of represents that the corresponding algorithm \( a \) can achieve the best results among all the algorithms in \( A \). From Figure 1, we can clearly observe that the curve of IHS lies strictly above the curves of AA and PLTS. This indicates that 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.

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 $\delta$ (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 $\gamma$. Due to the budget constraint of BMCP, a too large value of $\gamma$ will always guide IHS to infeasible spaces, which is obviously unsuitable. For this experiment, $\gamma$ ranges from 1 to 5 with a step size of 1. We run IHS with these five $\gamma$ 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 $\gamma$ 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: $t \to f$ 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 3). The target value of these two instances is set to 111000 and 109400, respectively. For the experiment, we run IHS and

![Figure 2: Performances of the IHS algorithm with different $\gamma$ values in terms of the $f_{avg}$ values (left) and the $t_{avg}$ values (right).](image-url)
Table 4
Comparison of the IHS algorithm with (IHS) and without (IHS−) the hyperplane search strategy.

<table>
<thead>
<tr>
<th>Instance</th>
<th>IHS</th>
<th>Std</th>
<th>IHS−</th>
<th>Std</th>
</tr>
</thead>
<tbody>
<tr>
<td>900_0.05_2000</td>
<td>7102</td>
<td>5.45</td>
<td>7107</td>
<td>5.45</td>
</tr>
<tr>
<td>500_0.05_2000</td>
<td>7102</td>
<td>5.45</td>
<td>7107</td>
<td>5.45</td>
</tr>
<tr>
<td>600_0.05_2000</td>
<td>6873</td>
<td>0</td>
<td>6873</td>
<td>0</td>
</tr>
<tr>
<td>700_0.05_2000</td>
<td>7104</td>
<td>0</td>
<td>7104</td>
<td>0</td>
</tr>
<tr>
<td>800_0.05_2000</td>
<td>7636</td>
<td>0</td>
<td>7636</td>
<td>0</td>
</tr>
<tr>
<td>900_0.05_2000</td>
<td>7058</td>
<td>0</td>
<td>7058</td>
<td>0</td>
</tr>
<tr>
<td>600_0.05_2000</td>
<td>67636</td>
<td>0</td>
<td>67636</td>
<td>0</td>
</tr>
<tr>
<td>700_0.05_2000</td>
<td>67636</td>
<td>0</td>
<td>67636</td>
<td>0</td>
</tr>
<tr>
<td>800_0.05_2000</td>
<td>7058</td>
<td>0</td>
<td>7058</td>
<td>0</td>
</tr>
<tr>
<td>900_0.05_2000</td>
<td>7058</td>
<td>0</td>
<td>7058</td>
<td>0</td>
</tr>
<tr>
<td>600_0.05_2000</td>
<td>67636</td>
<td>0</td>
<td>67636</td>
<td>0</td>
</tr>
<tr>
<td>700_0.05_2000</td>
<td>67636</td>
<td>0</td>
<td>67636</td>
<td>0</td>
</tr>
<tr>
<td>800_0.05_2000</td>
<td>7058</td>
<td>0</td>
<td>7058</td>
<td>0</td>
</tr>
<tr>
<td>900_0.05_2000</td>
<td>7058</td>
<td>0</td>
<td>7058</td>
<td>0</td>
</tr>
<tr>
<td>600_0.05_2000</td>
<td>67636</td>
<td>0</td>
<td>67636</td>
<td>0</td>
</tr>
<tr>
<td>700_0.05_2000</td>
<td>67636</td>
<td>0</td>
<td>67636</td>
<td>0</td>
</tr>
<tr>
<td>800_0.05_2000</td>
<td>7058</td>
<td>0</td>
<td>7058</td>
<td>0</td>
</tr>
<tr>
<td>900_0.05_2000</td>
<td>7058</td>
<td>0</td>
<td>7058</td>
<td>0</td>
</tr>
</tbody>
</table>

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.

PLTS 100 times independently with the default experimental settings. Each run is terminated immediately when the given target value is reached. The corresponding running time is served as the output of the experiment. The results and Y axes shows the running time and the cumulative probability to reach the given target, respectively.
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


