Probability learning based tabu search for the Budgeted Maximum Coverage Problem

Liwen Li[a], Zequn Wei[b], Jin-Kao Hao[b], Kun He[a,*]

[a]*School of Computer Science, Huazhong University of Science and Technology, Wuhan 430074, China*
[b]*LERIA, Université d'Angers, 2 bd Lavoisier, 49045 Angers, Cedex 01, France*

Expert Systems with Applications, 2021
https://doi.org/10.1016/j.eswa.2021.115310

Abstract

The Budgeted Maximum Coverage Problem (BMCP) is a general model with a number of real-world applications. Given n elements with nonnegative profits, m subsets of elements with nonnegative weights and a total budget, the BMCP aims to select some subsets such that the total weight of the selected subsets does not exceed the budget, while the total profit of the associated elements is maximized. BMCP is NP-hard and thus computationally challenging. We investigate for the first time an effective practical algorithm for solving this problem, which combines reinforcement learning and local search. The algorithm iterates through two distinct phases, namely a tabu search phase and a probability learning based perturbation phase. To assess the effectiveness of the proposed algorithm, we show computational results on a set of 30 benchmark instances introduced in this paper and present comparative studies with respect to the approximation algorithm, the genetic algorithm and the CPLEX solver.

Keywords: Budgeted maximum coverage problem; Learning-based optimization; Tabu search; Combinatorial optimization.

1. Introduction

We address the challenging Budgeted Maximum Coverage Problem (BMCP) (Khuller et al., 1999), which is a natural extension of the standard 0-1 Knapsack Problem (Kellerer et al., 2004) and the set cover problem (Balas and Padberg, 1972). Let $E = \{1, 2, ..., n\}$ be a set of n elements where each element $j \in E$ has a nonnegative profit $p_j > 0$ and $\mathcal{I} = \{1, 2, ..., m\}$ be a collection of m sets (or items) where each set (item) $i \in \mathcal{I}$ has a nonnegative weight $w_i > 0$ and covers a subset of elements $E_i \subseteq E$ determined by a relationship matrix $\mathbf{M}$. Given a

*Corresponding author.
Email address: brooklet60@hust.edu.cn (Kun He)
The first two authors contributed equally to this work.

budget (a knapsack with capacity) $C > 0$, the BMCP aims to select a subset $\mathcal{S} \subseteq \mathcal{I}$ such that the total weight $W(\mathcal{S})$ of the selected sets does not exceed the budget and the total profit $P(\mathcal{S})$ of the covered elements is maximized. Note that for a subset $\mathcal{S} \subseteq \mathcal{I}$, the profit p_j of an element j is counted only once even if the element may belong to multiple selected sets.

Formally, the BMCP can be stated as follows.

$$\begin{aligned} &\text{Maximize} \quad P(\mathcal{S}) = \textstyle\sum_{j \in \cup_{i \in \mathcal{S}} E_i} p_j \\ &s.t. \quad W(\mathcal{S}) = \textstyle\sum_{i \in \mathcal{S}} w_i \leqslant C, \mathcal{S} \subseteq \mathcal{I} \end{aligned}$$

The BMCP is closely related to the NP-hard set cover problem and 0-1 knapsack problem. If each set has a unit cost, the BMCP corresponds to a variant of the weighted set cover problem (Nemhauser et al., 1978) of picking K sets such that the total profit of the covered elements is maximized. Further, if each element also has a unit profit, the BMCP corresponds to a variant of the set cover problem (Hochba, 1997) of picking K sets such that the total number of covered elements is maximized. In a very simplified case where the sets and elements have a one-to-one mapping, the BMCP degenerates into the NP-hard 0-1 knapsack problem with weighted costs and profits.

As a generalized form of the NP-hard set cover problem and the NP-hard 0-1 knapsack problem, the BMCP is not only theoretically interesting, but also valuable for various applications, such as service provider operations, location of network monitors (Suh et al., 2006), news recommendation systems (Li et al., 2011), worker employment, financial decision making, software package installation, facility location (Khuller et al., 1999), and hybrid Software Defined Network (SDN) optimization (Kar et al., 2016). However, the BMCP was mainly investigated in terms of approximation algorithms. In particular, a $(1 - 1/e)$-approximation greedy algorithm was proposed in (Khuller et al., 1999), which is also the best possible approximation unless NP $\subseteq$ DTIME($n^{\mathrm{O}(\mathrm{log}\mathrm{log}n)}$). Given its high complexity, it is clear that such an algorithm has limited interest for practical applications. Since then, several approximation algorithms have been proposed for extensions of BMCP (Cohen and Katzir, 2008; Piva, 2019; van Heuven van Staereling et al., 2016), but little work has been done on the heuristic or meta-heuristic side. In 2016, Kar et al. formulated the hybrid Software Defined Network (SDN) optimization problems as a variant of BMCP, and proposed two greedy heuristic algorithms (Kar et al., 2016).

In this work, we aim to fill the gap by designing an effective practical algorithm for solving the BMCP. Our main contributions are summarized as follows.

- First, we introduce a mathematical model of the BMCP and propose an efficient practical algorithm for the problem that combines tabu search based local optimization and reinforcement learning based perturbation to reach a suitable balance between search intensification and diversification.

- Second, we introduce a set of 30 benchmark instances with varied properties and report computational results obtained with the proposed algorithm. These would be useful for future work on the BMCP, e.g., to evaluate the performance of other algorithms for this problem.

- Third, we investigate for the first time the general CPLEX solver for solving the BMCP and compare its results with those from the proposed algorithm. For the performance assessment, we also apply an approximation algorithm and a genetic algorithm for solving the BMCP. Experimental results demonstrate that our approach outperforms the general CPLEX solver and the two reference algorithms.

The rest of the paper is organized as follows. In Section 2, we provide a formal definition of the problem and illustrate its possible applications, followed by a review of related work in Section 3. In Section 4, we introduce the general framework and the composing ingredients of the probability learning based tabu search. Section 5 shows computational results and comparisons of PLTS with the approximation algorithm, the genetic algorithm and the CPLEX solver. We also analyze the parameters and carry out an ablation study to show the effectiveness of the probability learning based perturbation strategy in Section 6. Section 7 concludes the work.

2. Budgeted Maximum Coverage Problem

2.1. Problem Definition and Formulation

We formulate the Budgeted Maximum Coverage Problem (BMCP) using a 0/1 integer linear programming model, which is also suitable for the general Integer Linear Programming (ILP) solver CPLEX.

Given a collection of sets (items) $\mathcal{I} = \{1, 2, ..., m\}$ where each set (item) $i \in \mathcal{I}$ has a weight $w_i > 0$, and a set of elements $E = \{1, 2, ..., n\}$ where each element $j \in E$ has a profit $p_j > 0$, we are asked to select a subset of sets $\mathcal{S} \subseteq \mathcal{I}$ such that the total weight $W(\mathcal{S})$ of the selected sets does not exceed the budget (knapsack capacity) C and the total profit $P(\mathcal{S})$ of their covered elements is maximized.

Let y_i $(i = 1, 2, ..., m)$ be a binary variable such that $y_i = 1$ if set i is selected, and $y_i = 0$ otherwise. Let $\mathbf{M}$ be an $m \times n$ binary relationship matrix between m sets and n elements where $\mathbf{M}_{ij} = 1$ indicates the presence of element j in set i. For each element j $(j = 1, 2, ..., n)$, define $H_j = \sum_{i=1}^{m} y_i \mathbf{M}_{ij}$ that counts the number of appearances of element j in the sets. Let x_j be a binary variable such that $x_j = 1$ if $H_j > 0$, and $x_j = 0$ otherwise.

The BMCP can be formulated as the following integer linear program:

$$\text{Maximize} \quad P(\mathcal{S}) = \sum_{j=1}^{n} x_j p_j$$

$$s.t. \quad (1) \quad W(\mathcal{S}) = \sum_{i=1}^{m} y_i w_i \leqslant C$$

$$(2) \quad y_i \in \{0, 1\}, \ i = 1, \ldots, m$$

$$(3) \quad H_j = \sum_{i=1}^{m} y_i \mathbf{M}_{ij}, \ j = 1, \ldots, n$$

$$(4)\quad x_j = \begin{cases} 1, & if\ H_j > 0; \\ 0, & \text{otherwise.} \end{cases}$$

2.2. Possible Applications

The BMCP exhibits a wide range of real-world applications. To name only a few, we provide the following application scenarios.

- **Software package installation**. Assume that a computer server has a certain storage capacity, and it needs to install some application software packages, each of which has a certain profit and needs to install some dependent packages in advance, which occupy a certain amount of memory space. An important decision is which set of packages should be installed to maximize the software profits without exceeding the server storage capacity limit. In the BMCP model of this application, each set corresponds to a package and each element corresponds to a software package. The weight of each set equals the amount of memory required for the corresponding package, and the profit of each element equals the profit of each software.

- **Service provider operations**. The budget corresponds to the budget of a service provider, and each element corresponds to a service that can be carried out in a city. Each service has a profit, but needs to establish a facility beforehand, for example the base station, for the service. The service can be supported by any of the associated facilities, and each facility needs a certain building cost. The service provider needs to determine which subset of facilities to be built under the budget so as to earn a maximized profit on the associated services.

- **Worker employment**. A company has an employment budget for workers. Each worker has several skills, and each skill has a priority score basing on its importance to the company if covered by at least one employed worker. Thus, a set corresponds to a worker and an element corresponds to a skill. The goal is to employ workers under the total budget such that the required total skill score for the company is maximized.

- **Financial decision making**. The budget corresponds to a company's project investment budget. Each set corresponds to an investment leader, together with a certain employment cost and a variety of projects to be invested. Each element corresponds to a project, together with a certain profit if invested. The goal is to hire a subset of the project leaders under the total budget to maximize the total profit of the associated projects.

- **Program assignment**. The programmers have a total budget of working time, and each program writing takes a certain amount of time. So each program corresponds to a set with a time cost. A task is related to several programs and we could earn a profit if any one of the associated programs is completed. We need to assign a subset of programs to the programmers under the total budget time so that the total profit is maximized.

3. Related Work

For related work, we discuss the BMCP and its related problems as well as approaches combining heuristics with learning techniques.

3.1. Existing work for the BMCP

The BMCP is studied theoretically on approximation algorithms early in 1999 (Khuller et al., 1999), in which the authors presented a $(1-1/e)$-approximation algorithm, and claimed it is the best possible approximation unless NP $\subseteq$ DTIME($n^{\mathrm{O}(\log\log n)}$). Followup works are on approximation algorithms for extension or generalized versions of the BMCP (Cohen and Katzir, 2008; Piva, 2019; van Heuven van Staereling et al., 2016). To the best of our knowledge, little work is done from the perspective of heuristics or meta-heuristics for the BMCP. Recently in 2016, Kar et al. formulated the maximum coverage minimum cost hybrid Software Defined Network (SDN) optimization problems both for path coverage and hop coverage, and proposed two efficient greedy heuristic algorithms, the maximum number of uncovered paths first and maximum number of minimum hop covered paths first for the BMCP variant problem (Kar et al., 2016). Given the high complexity of the BMCP, the approximation of $(1 - 1/e)$ has limited interest for practical applications, and it is valuable to design efficient and effective algorithms.

We observe that the existing studies focused on the theoretical aspects of the BMCP, while ignoring the practical solution approaches. This work fills this gap by presenting an efficient metaheuristic algorithm for solving the BMCP.

3.2. The BMCP Related Problems

As mentioned in Section 1, the BMCP degenerates to the well-known set cover problem (Balas and Padberg, 1972) when the weight w_i, $i \in \mathcal{I}$ and profit p_j, $j \in E$ are all set to 1. In such a case the goal of BMCP reduces to cover as many elements as possible. The BMCP can also be reduced to the standard NP-hard 0-1 Knapsack Problem (Kellerer et al., 2004) when $m = n$ and each set $i \in \mathcal{I}$ covers exactly one element $i \in E$. As a generalization problem of the two typical NP-hard problems, the BMCP is computationally challenging.

Moreover, the BMCP is highly related to the Set-Union Knapsack Problem (SUKP) (Goldschmidt et al., 1994). In the SUKP, each set (item) i has a nonnegative profit p_i and each element j has a nonnegative weight w_j. The goal of SUKP is to package a subset $\mathcal{S}$ of sets (items) in order to maximize the total profit $P(\mathcal{S})$ of the selected sets (items), while the total weight $W(\mathcal{S})$ of the covered elements does not exceed the knapsack capacity C.

The SUKP can be formulated as follows:

$$\begin{aligned} \text{Maximize} \quad & P(\mathcal{S}) = \textstyle\sum_{j\in\mathcal{S}} p_j \\ s.t. \quad & W(\mathcal{S}) = \textstyle\sum_{i\in\cup_{j\in\mathcal{S}} E_j} w_i \leqslant C. \end{aligned}$$

The BMCP swaps the attributes of sets (items) and elements (see Figure 1), and thus we call BMCP the "dual" problem of SUKP, and SUKP the "dual" problem of BMCP. The SUKP has received increasing attention in recent years.

Early in 1994, Goldschmidt et al. first presented an exact algorithm based on dynamic programming to solve the SUKP (Goldschmidt et al., 1994). In 2014, Arulselvan presented a greedy strategy based on an approximation algorithm (Arulselvan, 2014). Recently in 2016, Taylor designed an approximation algorithm using results of the related densest k-subhypergraph problem (Taylor, 2016). Then He et al. developed a binary artificial bee colony algorithm for the SUKP in 2018 (He et al., 2018). In 2019, Wei et al. proposed an iterated two-phase local search called I2PLS (Wei and Hao, 2019), and Geng et al. presented a hybrid binary particle swarm optimization with tabu search (Lin et al., 2019). In 2020, He et al. proposed a hybrid Jaya algorithm for solving the SUKP (Wu and He, 2020). Very recently, Wei et al. introduced two heuristic algorithms and achieved remarkable results on the SUKP (Wei and Hao, 2020, 2021).

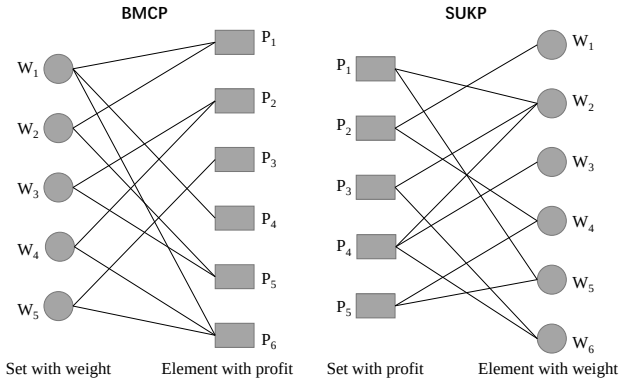

Fig. 1. Relationship of the BMCP and the SUKP.

For the two highly related knapsack problems, the SUKP has received a lot of attention in recent years, while the BMCP has been overlooked. It would be meaningful to develop the mathematical model and solution approaches for the BMCP.

Furthermore, there are some other extensions that are related to the BMCP, such as the generalized maximum coverage problem (Cohen and Katzir, 2008), wherein the authors used a variation of greedy algorithm to settle an extension of the BMCP; the maximum coverage problem with group budget constraint (Chekuri and Kumar, 2004), wherein the authors studied a variant of the BMCP that labeled the maximum coverage problem with group budget constraints; the budgeted maximum coverage with overlapping costs (Curtis et al., 2010), wherein the authors modeled the problem of monitoring a listserv as a type of the BMCP; the ground-set-cost budgeted maximum coverage problem (van Heuven van Staereling et al., 2016), which is a natural variant of the BMCP, etc.

3.3. Combining Learning with Heuristics

In recent years, researches on combining heuristics with learning techniques have received increasing attention. In 2001, Boyan and Moore proposed a learn-

ing evaluation function to improve the optimization by local search (Boyan and Moore, 2000). In 2016, Zhou et al. introduced a reinforcement learning based local search (RLS) for solving grouping problems (Zhou et al., 2016). In 2017, Wang and Tang presented a machine-learning based memetic algorithm for the multi-objective permutation flowshop scheduling problem (Wang and Tang, 2017). And Benlic et al. proposed a hybrid breakout local search and reinforcement learning approach to the vertex separator problem (Benlic et al., 2017). In 2020, Wang et al. combined local search and reinforcement learning for the minimum weight independent dominating set problem (Wang et al., 2020).

In this work, we present a probability learning based local search to address the BMCP. We are interested in investigating a probabilistic guided local search method for the BMCP that adopts learning techniques to process the information collected from the search process so as to improve the heuristic performance.

4. The Proposed Algorithm

This section describes the proposed probability learning based tabu search (PLTS) algorithm for solving the BMCP. The overall framework is introduced first, followed by detailed algorithm description.

4.1. General Framework

The proposed PLTS algorithm is mainly inspired by the studies of (Zhou et al., 2016, 2018). By combining probability learning technique with tabu search, the PLTS algorithm is composed of two complementary search phases: a descent-based tabu search procedure to find new local optimal solutions and a local optimal perturbation procedure based on probability learning instruction (Algorithm 1).

Algorithm 1 Probability Learning Based Tabu Search

Input: Instance A, time limit T_{max}, probability vector P, *flip* neighborhood N_1 and *swap* neighborhood N_2, tabu search depth α_{max}
Output: The best solution found $\mathcal{S}^*$
// Initialization of solution $\mathcal{S}_0$, §4.3
$\mathcal{S}_0 \leftarrow Initial\_Solution(A)$
$\mathcal{S}^* \leftarrow \mathcal{S}_0$
while $RunningTime \leqslant T_{max}$ **do**
  $P_0 \leftarrow Initial\_Probability\_Vector(P)$
  // Optimization of the first search phase, §4.4
  $(\mathcal{S}_b, P) \leftarrow$Tabu_Search$(\mathcal{S}_0, N_1, N_2, P_0, \alpha_{max})$
  if $f(\mathcal{S}_b) > f(\mathcal{S}^*)$ **then**
    $\mathcal{S}^* \leftarrow \mathcal{S}_b$ // Update the best solution $\mathcal{S}^*$ found so far
  end if
  // Optimization of the second search phase, §4.5
  $\mathcal{S}_0 \leftarrow Probability\_Perturbation(\mathcal{S}_b, P)$
end while
return $\mathcal{S}^*$

Specifically, the PLTS algorithm is randomly initialized and then a descent-based local search is applied to reach a local optimal as the initial solution. Then a tabu search procedure is adopted to explore a new local optimal solution within the *flip* neighborhood N_1 and the *swap* neighborhood N_2 (Section 4.4.1). We update the probability vector whenever a better solution is found (Section 4.4.2). Specifically, if a set is selected to join the candidate set group, we increase its value of the probability vector as the reward. Otherwise, if a set is removed from the candidate set group, we reduce its probability as the punishment. When the tabu search is exhausted, the PLTS algorithm applies a probability learning based perturbation to guide the search to unexplored regions. In Section 4.5, we use the probability vector to randomly generate a new solution and start the tabu search again from this new solution. During this search process, the best solution found is recorded and returned as the final output within the time limit.

4.2. Search Space and Evaluation Function

The search space Ω explored through the tabu search process depends on the number of sets in a problem instance. Given a BMCP instance with m sets $\mathcal{I} = \{1, 2, ..., m\}$ and n elements $E = \{1, 2, ..., n\}$ where each set $i(i = 1, ..., m)$ corresponds to a subset of elements $E_i \subseteq E$, a candidate solution $\mathcal{S}$ of Ω can be represented by $\mathcal{S} = \langle V, \bar{V} \rangle$ where V represents the set of selected sets and $\bar{V}$ represents the remaining sets. The search space Ω can be represented as follows:

$$\Omega = \{(x_1, x_2, ..., x_m) | x_i \in \{0, 1\}, 1 \leqslant i \leqslant m\}. \tag{1}$$

For an arbitrary solution $\mathcal{S} \subseteq \Omega$, the total weight of $\mathcal{S}$ is:

$$W(\mathcal{S}) = \sum_{i \in \mathcal{S}} w_i. \tag{2}$$

The evaluation function $f(\mathcal{S})$ that corresponds to the total profit of $\mathcal{S}$ is defined as:

$$f(\mathcal{S}) = \sum_{j \in \cup_{i \in \mathcal{S}} E_i} p_j. \tag{3}$$

Given an instance with budget C, the purpose of PLTS is to find a solution $\mathcal{S}$ while the total weight of picked sets $W(\mathcal{S}) \leqslant C$ and the objective value $f(\mathcal{S})$ is as large as possible.

4.3. Initialization

The PLTS algorithm starts the search from an initial solution (Algorithm 2). For simplicity, we employ a simple and fast descent-based local search procedure to generate a good initial solution which is carried out in two steps. First of all, we pick some sets from the candidate group randomly until the budget constraint is reached. Then we adopt a simple descent-based algorithm to exchange one selected set with one unselected set, which we call an *action*. If an action

increases the total profit of the covered elements and the total weight does not exceed the budget, then we select that move. At the end of this process, we obtain a local optimal feasible solution which is used as the initial solution for the tabu search procedure.

Algorithm 2 Generating the Initial Solution

```
1: Function Initial_Solution()
2: Input: Instance A
3: Output: Initial solution S_0 = (x_1, x_2, ..., x_m)
4: while TotalWeight ⩽ C do
5:    Randomly add an unselected set i into the set group.
6:    if TotalWeight+w_i ⩽ C then
7:       x_i ← 1
8:    else
9:       break
10:   end if
11: end while
12: S_0 ← (x_1, x_2, ..., x_m)
13: S_0 ← Descent_based_Local_Search(S_0)
14: return S_0
```

4.4. Tabu Search for Solution Improvement

The descent-based local search can quickly find a local optimum. However, the quality of this initial solution usually can be further improved. In particular, tabu search is known as one of the most popular local search methods for several knapsack problems (Glover and Laguna, 1997). We take this local optimum as the input solution of the tabu search (TS) procedure (Algorithm 3) to find better solutions.

The tabu search procedure examines two neighborhoods N_1 and N_2 (Section 4.4.1) simultaneously to explore candidate solutions. As shown in Algorithm 3, at each iteration, TS picks the best feasible neighboring solution $\mathcal{S}' \in (N_1(\mathcal{S}) \bigcup N_2(\mathcal{S}))$ according to the evaluation function f given by Equation 3 such that $\mathcal{S}'$ is the best solution not forbidden by the tabu list. If no improving solution exists in $N_1(\mathcal{S}) \bigcup N_2(\mathcal{S})$, the tabu search process selects the best solution $\mathcal{S}'$ from the candidate neighboring solutions even if $f(\mathcal{S}') < f(\mathcal{S})$. This feature allows the tabu search to go beyond the local optimum.

To avoid revisiting already encountered solutions during the search, we employ a tabu list to record the sets involved in the move operation. T_i is the tabu tenure (duration) of set i and is determined as follows:

$$T_i = 4 + max(m, n)/100. \tag{4}$$

where m is the number of sets and n is the number of elements. As a result, the length of tabu tenure will change with the size of different instances. Our preliminary experiment discloses that this adaptive strategy is helpful to keep the number of forbidden sets within a reasonable range.

During the tabu search, we also need to update the probability vector of the set simultaneously (Section 4.4.2). If a set is selected into the candidate group,

we will reward its probability, whereas if the set is taken out, we will reduce its probability as the punishment. This probability vector will be used during the perturbation procedure.

The tabu search process terminates when the number of iterations without improving $\mathcal{S}'$ reaches the tabu search depth α_{max}. Considering the fact that the search space will increase with the size of the instance, a larger search depth is beneficial for the algorithm to explore unvisited regions. On the contrary, PLTS can effectively explore the space with a smaller search depth for the instances of small sizes. In order to ensure the performance of the proposed algorithm on the BMCP instances of different sizes, we employ an adaptive mechanism to tune the tabu search depth, i.e., $\alpha_{max} = (1100 - m) \times 20$. Here m is the number of sets. Therefore, our algorithm can automatically choose the corresponding termination conditions for different instances. We present an analysis of the tabu search depth in Section 6.1.

Algorithm 3 Tabu Search Procedure

```
1: Function Tabu_Search()
2: Input: Input solution S, neighborhood N_1, N_2, probability vector P_0, tabu search depth
   α_max
3: Output: Best solution S_b found during the tabu search and probability vector P
4: S_b ← S                                          //S_b records the best solution found so far
5: α ← 0                    //α counts the number of consecutive non-improving iterations
6: while α < α_max do
7:    S ← argmax{f(S') : S' ∈ (N_1(S) ⋃ N_2(S))  and S' is not forbidden by the tabu list}
8:    // Update the probability vector, §4.4.2
      P ← probability_vector_updating(P_0)
9:    if f(S) > f(S_b) then
10:      S_b ← S                                  // Update the best solution S_b found so far
11:      α ← 0
12:   else
13:      α ← α + 1
14:   end if
15:   Tabu_list_updating()
16: end while
17: return S_b
```

4.4.1. Move Operators and Neighborhoods

The neighborhood used by $Tabu\_Search()$ consists of two basic neighborhoods, namely the *flip* neighborhood N_1 and the *swap* neighborhood N_2. For a current feasible solution indicated by $\mathcal{S} = (x_1, x_2, ..., x_m)$, the function of *flip* neighborhood N_1 is to flip the value of a variable x_q in $\mathcal{S}$ while satisfying the budget constraint C, that is, $Flip(q)$ changes the value of a variable x_q to its complementary value $1 - x_q$. Therefore, all possible solutions that can be obtained by the *flip* operator constitute the N_1 neighborhood of solution $\mathcal{S}$. $N_1(\mathcal{S})$ can be defined as follows:

$$N_1(\mathcal{S}) = \{\mathcal{S}' \mid \mathcal{S}' = \mathcal{S} \oplus Flip(q), q \in \mathcal{S}, \sum_{i \in \mathcal{S}'} w_i \leqslant C\}. \tag{5}$$

The second neighborhood N_2 is defined by the *swap* operator $Swap(p, q)$ where p is in the selected set group V and q is in the unselected set group $\bar{V}$. Note that the *swap* operator also needs to meet the budget constraint C. The *swap* neighborhood $N_2(\mathcal{S})$ can be defined as follows:

$$N_2(\mathcal{S}) = \{\mathcal{S}' \mid \mathcal{S}' = \mathcal{S} \oplus Swap(p, q), p \in V, q \in \bar{V}, \sum_{i \in \mathcal{S}'} w_i \leqslant C\}. \tag{6}$$

The tabu search algorithm explores the union of the two neighborhoods, $N(\mathcal{S}) = N_1(\mathcal{S}) \bigcup N_2(\mathcal{S})$, and N is bounded in size by $O(m + |V| \times |\bar{V}|)$.

4.4.2. Probability Update Policy

Our probability learning based tabu search algorithm borrows the idea of reinforcement learning in the area of machine learning. Reinforcement learning is defined as the concept of how an agent should take actions in an environment to maximize the cumulative rewards. The intuition is that actions leading to higher rewards are more likely to recur. In the BMCP, for each set there are two possible states, selected or unselected. There are also two possible actions (moves) for a set: putting into the candidate group or removing from the candidate group. Since there are numerous move operations during the tabu search process, it is beneficial to integrate some learning techniques to guide the search to update the probability vector.

We define a probability vector of length n, where p_i denotes the probability that set i is selected to be put into the candidate group. Initially, each of the probability values in the probability vector is set to 0.50, indicating that each set has a half chance to be selected into the candidate group.

During the tabu search procedure, if a set i is selected into the candidate group, we update its probability value as follows:

$$p_i(t + 1) = \beta + (1 - \beta) \times p_i(t), \tag{7}$$

where β $(0 < \beta < 1)$ is a reward factor. On contrast, if a set i is taken out of the candidate group, we punish its probability by a penalization factor γ $(0 < \gamma < 1)$:

$$p_i(t + 1) = (1 - \gamma) \times p_i(t). \tag{8}$$

Our probability update scheme is inspired by the learning automata (Narendra and Thathachar, 1989). The principle of this scheme is to increase the selection probability when sets are selected feasibly and reduce the selection probability when sets are taken out. The probability vector is updated during the tabu search procedure. And in the perturbation procedure (Section 4.5), we can generate new solutions directly based on the probability vector. We provide an analysis of the reward factor β and the penalization factor γ in Section 6.2.

4.5. Probability Learning based Perturbation

The purpose of the perturbation procedure is to diversify the search by exploring new search areas. The probability learning based perturbation plays an important role when the tabu search is exhausted. Specifically, each set will be dropped or picked according to the probability vector, which will generate a new perturbed solution as the starting point for the next round of tabu search. We consider and compare the following two perturbation strategies:

1) *Random perturbation*: For a feasible solution $\mathcal{S}$, this policy randomly selects half sets to be removed from the candidate group (regardless of its probability value), then uses the descent search algorithm to select sets until the candidate group reaches the budget. Note that this selection policy does not use any information gathered from the search history.

2) *Probability perturbation*: As shown in Algorithm 4, starting from the input local optimal solution $\mathcal{S}_b$, this policy first drops the selected sets in $\mathcal{S}_b$ according to the probability vector P. Then, we put unselected sets into the candidate group under the guidance of vector P. Specifically, for a randomly unselected set j, we set $x_j = 1$ according to the j-th value in probability vector P, when set j can bring a feasible solution after being added into the candidate group. This process iterates until the budget is reached. The new perturbed solution $\mathcal{S}_0$ will serve as a new input solution for the tabu search. Thus, the probability perturbation makes full utilization of the probability vector. If a set has a high probability of being selected, it has a higher probability of being taken out of the candidate group. On the contrary, if the value of the probability vector is small, it has a higher probability of being selected into the candidate group. This strategy enables the algorithm to explore new search areas from a feasible solution. Furthermore, we will show the impact of this perturbation strategy on the performance in Section 6.3.

5. Experimental Results

In this section, we present experimental results of the proposed PLTS algorithm on 30 benchmark instances that we designed for the BMCP, and show comparisons with the results obtained by the general CPLEX solver, as well as the typical approximation algorithm (Khuller et al., 1999) we implemented and a genetic meta-heuristic algorithm we designed as a baseline. Then we analyze the probability learning based perturbation of the PLTS algorithm.

5.1. Benchmark Instances

As there are no existing benchmark instances for the BMCP, inspired by the instances of SUKP (He et al., 2018), we generate 30 instances [2] with similar characteristics to the instances of SUKP. For the diversity of instances, we divide the instances into three sets, ranging from 585 to 1000, based on the relationship

[2] Available at: https://github.com/lly53/BMCP_instance.

Algorithm 4 Probability Perturbation Policy

```
Function Probability_Perturbation()
Input: Input solution S_b, number of sets (m), probability vector P, budget C
Output: New solution S_0 = (x_1, x_2, ..., x_m)
for each selected set i in S_b (x_i = 1) do
    p ← rand(0, 1)
    if p ⩽ p_i then
        x_i ← 0                                   //Drop set according to probability vector
        Update TotalWeight
    end if
end for
for each unselected set j in S_b (x_j = 0) do
    if TotalWeight + w_j ⩽ C then
        p ← rand(0, 1)
        if p > p_j then
            x_j ← 1                               //Pick set according to probability vector
            Update TotalWeight
        end if
    else
        break
    end if
end for
S_0 ← (x_1, x_2, ..., x_m)
return S_0
```

between the number of sets and the number of elements. The number of sets in the first group is less than the number of elements. In the second group, the number of sets equals the number of elements, and in the third group the number of sets is greater than the number of elements. Let $\mathbf{M}$ be an $m \times n$ binary relation matrix between m sets and n elements where $\mathbf{M}_{ij} = 1$ indicates the presence of element j in set i. Experiments show that the budget has a strong correlation with the density of relation matrix. If the backpack budget is too large and the relation matrix density is too high, it is easy for all elements to be covered, resulting in a trivial solution to this problem. Therefore, to avoid the number selection of sets that easily cover all elements, we adjusted the density of the relationship matrix $\mathbf{M}$ to a fixed value according to the budget. When the budget is 2000, the density of the relationship matrix is 0.05. When the budget is 1500, the density of the relationship matrix is 0.075. Then *bmcp_m_n_α_C* designates an instance with m sets and n elements, density of relationship matrix α and budget C, where $\alpha = (\sum_{i=1}^{m} \sum_{j=1}^{n} M_{ij})/(mn)$.

5.2. Experimental Setup

The proposed PLTS algorithm was coded in C++ and compiled using the g++ compiler with the -O3 option. The experiments were carried on an Intel Xeon E5-2670 processor with 2.5 GHz and 2 GB RAM under the Linux operating system.

Table 1 shows the description and setting of the parameters used for our experiments. To obtain the experimental results, each instance was solved 30

Table 1: Parameter setup.

Parameters	Section	Description	Value
T_{max}	4.1	time limit	600
β	4.4.2	reward factor	0.50
γ	4.4.2	penalization factor	0.50

times independently with different random seeds, each run being limited to 600 seconds.

Since there is no result reported by using the general integer linear programming (ILP) approach for solving the BMCP, we present computational results of the CPLEX solver (version 12.8) under the time limit of 2 hours as well as 5 hours for each instance based on the 0/1 integer linear programming model.

To complete the computational assessment of the proposed algorithm, we also adopt two other reference algorithms, i.e., the traditional approximation algorithm for the BMCP as introduced in (Khuller et al., 1999) and a genetic algorithm. For the genetic algorithm, we first initialize the population (of size 10) randomly and employ the common backbone-based crossover to create new offspring solutions. Then a descent procedure is adopted to improve the obtained solution. To make a fair comparison, we run the genetic algorithm on our computing platform 30 times with the same cut-off time (600 seconds).

5.3. Comparison on Computational Results

We first assess the performance of the proposed PLTS algorithm with respect to the CPLEX solver and the two reference algorithms. In Tables 2-4 we report computational results on the three sets of benchmark instances. The first column indicates each instance name, the second and third columns list the results of the approximate algorithm (AA) and the best result obtained by the genetic algorithm (GA), respectively, followed by the best lower bound (LB) and upper bound (UB) achieved by CPLEX. To thoroughly compare the experimental results, we ran CPLEX for 2 hours and 5 hours respectively for each instance. Then, f_{best} indicates the best objective value obtained by PLTS over 30 runs, followed by the average value (f_{avg}), standard deviations (Std), and average running time (t_{avg}) in seconds. As can be seen from Tables 2-4, the PLTS algorithm we proposed is always superior to the traditional approximation algorithm, the genetic algorithm and the lower bounds of the CPLEX solver. Moreover, we carry out the non-parametric Wilcoxon signed-rank test to check the statistical difference between PLTS and the reference algorithms (AA and GA). The small *p-values* (2.88e-06 for PLTS vs. AA and 1.73e-06 for PLTS vs. GA) clearly disclose that there are significant differences between PLTS and its competitors.

In Figure 2, we further illustrate the comparative results of our PLTS algorithm with the three baselines. This figure shows the experimental results of the approximate algorithm, the genetic algorithm and the lower bounds obtained by CPLEX solver of 2 hours and 5 hours and the best results obtained by the

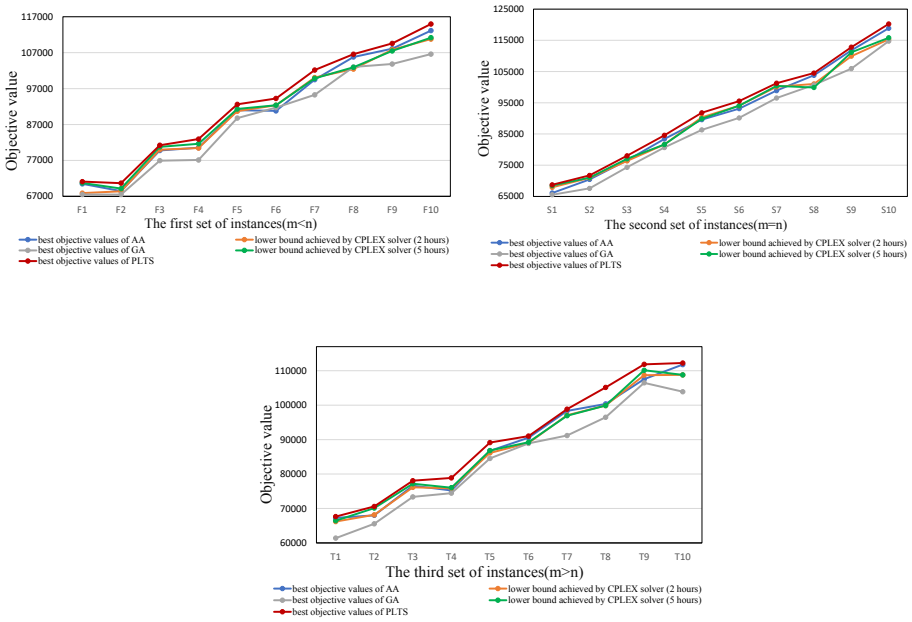

Fig. 2. Comparisons of our PLTS algorithm with the approximation algorithm, the genetic algorithm and the lower bounds of CPLEX (2 hours and 5 hours) on the three sets of BMCP instances.

Table 2: Comparison of PLTS with the approximation algorithm, the genetic algorithm and the CPLEX solver on the first set of instances ($m < n$).

Instance	AA	GA	CPLEX (2 hours)		CPLEX (5 hours)		PLTS			
			LB	UB	LB	UB	f_{best}	f_{avg}	Std	t_{avg}
bmcp_585_600_0.05_2000	70494	67517	67910	73495.88	70742	74224.94	**71102**	71065.17	82.36	309.602
bmcp_585_600_0.075_1500	68475	67514	68418	77549.43	69172	76716.90	**70677**	70677.00	0.00	61.242
bmcp_685_700_0.05_2000	79778	76938	79997	88954.29	80783	88447.67	**81227**	80585.73	508.37	522.060
bmcp_685_700_0.075_1500	80457	77133	80443	92328.30	81639	91378.98	**82955**	82951.40	19.39	109.670
bmcp_785_800_0.05_2000	90975	88787	90705	102198.73	91319	101585.69	**92608**	92587.60	34.30	252.589
bmcp_785_800_0.075_1500	90786	91711	92358	107354.51	92358	106842.67	**94245**	94245.00	0.00	248.128
bmcp_885_900_0.05_2000	99498	95255	100085	114313.81	99845	113746.97	**102162**	101331.53	174.95	206.025
bmcp_885_900_0.075_1500	105793	103018	102423	122684.38	102933	122093.51	**106577**	105942.43	334.18	489.396
bmcp_985_1000_0.05_2000	108105	103844	107820	125903.67	107488	124487.37	**109567**	109408.77	227.85	212.193
bmcp_985_1000_0.075_1500	113137	106604	110769	134440.36	111177	133789.78	**114969**	113838.07	509.34	485.677

PLTS algorithm. The $X-axis$ in each subfigure indicates the 10 instances of each set, and the $Y-axis$ shows the objective values of the compared algorithms. To facilitate the presentation of the results in Figure 2, the three sets of instances are denoted by F1−F10 ($m < n$), S1−S10 ($m = n$) and T1−T10 ($m > n$), respectively. Figure 2 indicates that the proposed PLTS algorithm outperforms the baseline algorithms on all instances. For all the 30 instances,

Table 3: Comparison of PLTS with the approximation algorithm, the genetic algorithm and the CPLEX solver on the second set of instances ($m = n$).

Instance	AA	GA	CPLEX (2 hours)		CPLEX (5 hours)		PLTS			
			LB	UB	LB	UB	f_{best}	f_{avg}	Std	t_{avg}
bmcp_600_600_0.05_2000	66095	65546	67917	73495.08	68477	72880.93	**68738**	68472.00	71.09	95.638
bmcp_600_600_0.075_1500	70445	67581	70947	77379.97	71018	76337.67	**71746**	71746.00	0.00	27.975
bmcp_700_700_0.05_2000	76552	74320	76367	85587.33	77056	84855.44	**78028**	77859.27	75.16	127.445
bmcp_700_700_0.075_1500	83400	80679	81645	93026.66	81645	92151.99	**84576**	84375.70	550.91	196.995
bmcp_800_800_0.05_2000	89582	86322	90344	101141.03	89872	100373.77	**91795**	91576.27	309.05	307.274
bmcp_800_800_0.075_1500	93115	90157	94049	108713.00	94049	108005.62	**95533**	95509.60	70.20	239.146
bmcp_900_900_0.05_2000	98893	96523	100108	115682.55	100412	114551.09	**101265**	101231.17	62.94	325.683
bmcp_900_900_0.075_1500	103795	100715	101035	120452.83	99888	118554.77	**104521**	104521.00	0.00	176.865
bmcp_1000_1000_0.05_2000	111786	105941	109928	131194.18	111155	128583.63	**112802**	111897.07	636.78	577.668
bmcp_1000_1000_0.075_1500	118869	114763	115313	139152.89	115824	137900.40	**120246**	118467.87	546.67	279.220

Table 4: Comparison of PLTS with the approximation algorithm, the genetic algorithm and the CPLEX solver on the third set of instances ($m > n$).

Instance	AA	GA	CPLEX (2 hours)		CPLEX (5 hours)		PLTS			
			LB	UB	LB	UB	f_{best}	f_{avg}	Std	t_{avg}
bmcp_600_585_0.05_2000	67256	61381	66184	71739.68	66452	71094.27	**67636**	67460.80	350.40	202.660
bmcp_600_585_0.075_1500	68005	65545	68145	77321.51	70113	76332.85	**70588**	70406.63	105.09	584.205
bmcp_700_685_0.05_2000	76600	73354	76139	83561.86	77176	82815.11	**78054**	78037.00	51.00	197.590
bmcp_700_685_0.075_1500	75224	74451	75841	86956.88	76033	86566.79	**78869**	78869.00	0.00	46.987
bmcp_800_785_0.05_2000	86750	84528	86139	97891.07	86813	97477.34	**89138**	88581.20	103.40	204.141
bmcp_800_785_0.075_1500	90548	88902	89018	103644.12	89229	102867.98	**91021**	91010.20	25.67	297.211
bmcp_900_885_0.05_2000	98337	91187	97088	110828.83	96945	109948.52	**98840**	98718.00	151.34	227.976
bmcp_900_885_0.075_1500	100359	96475	99881	119045.52	99888	118554.77	**105141**	104397.93	691.61	229.644
bmcp_1000_985_0.05_2000	107548	106497	108714	126779.11	110134	125574.84	**111859**	111228.80	828.72	244.920
bmcp_1000_985_0.075_1500	111778	99913	108801	131873.01	108801	130981.38	**112250**	112125.87	143.22	234.614

the best solutions (f_{best}) obtained by our PLTS algorithm are better than the results obtained by the approximation algorithm, the genetic algorithm and the lower bounds (LB) obtained by CPLEX solver no matter for 2 hours or 5 hours. This figure again confirms the performance of the proposed algorithm.

6. Analysis

6.1. Analysis of Tabu Search Depth

As introduced in Section 4.4, the PLTS algorithm employs an adaptive mechanism to adjust the tabu depth during the search. Here we analyze the influence of tabu search depth α_{max} on the performance of the proposed algorithm. This experiment is based on a selection of eight representative instances, as listed in Table 5. For each instance, we ran our PLTS algorithm 30 times with the same cut-off time as in Section 5.2 and recorded the best objective value f_{best}. In this

experiment, the tabu search depth α_{max} takes its values with $(1100 - m) \times 10$ ($TS \times 10$), $(1100 - m) \times 20$ ($TS \times 20$) and $(1100 - m) \times 30$ ($TS \times 30$). In addition, we also tested the α_{max} from 1000 to 10000 with a step size of 1000.

Table 5 shows the experimental results of this analysis. The first row gives the settings of α_{max} (Tabu depth) and the first column shows the instances tested. The f_{best} values of each setting of α_{max} are shown in rows 2 to 9, respectively. The last row presents the average values of each column. The best results of the compared settings are highlighted in bold.

Table 5 discloses that PLTS obtains better results when $\alpha_{max} = (1100 - m) \times 20$. This justifies the adopted setting of α_{max}. Moreover, we carried out the non-parametric Friedman test to compare the statistical significance of the results in Table 5. The small *p-value* (1.61e-4 < 0.05) indicates that the performance differences among the settings of α_{max} are statistically significant.

Table 5: Analyze the influence of tabu search depth α_{max} on the performance of the PLTS algorithm.

InstanceTabu depth	1000	2000	3000	4000	5000	6000	7000	8000	9000	10000	$TS \times 10$	$TS \times 20$	$TS \times 30$
bmcp_600_600_0.05_2000	68453	68609	68487	68738	68738	68738	68453	68453	68738	68738	68738	68738	68738
bmcp_600_585_0.075_1500	70473	70588	70588	70588	70318	70588	70318	70588	70318	70588	70588	70588	70588
bmcp_685_700_0.05_2000	80965	81204	81227	81227	81227	81227	80197	81227	81227	81227	81227	81227	81227
bmcp_800_785_0.05_2000	88562	88562	88863	88562	88562	89007	88562	89084	89138	88562	89084	89138	88562
bmcp_885_900_0.05_2000	101745	102000	101793	102000	102162	102071	102071	102071	102162	102071	102155	102162	102071
bmcp_900_885_0.075_1500	105076	105141	105141	105141	105141	105141	104962	105141	105141	105141	105141	105141	105044
bmcp_1000_1000_0.075_1500	120217	118959	120246	119620	119924	119594	118226	119412	119437	120217	119168	120246	120246
bmcp_1000_985_0.05_2000	111568	111859	111821	111821	111859	111859	111407	111859	111859	111821	111859	111859	111612
Average value	93382.38	93365.25	93520.75	93462.13	93491.38	93528.13	93024.50	93479.38	93502.50	93545.63	93495	**93637.38**	93511

6.2. Analysis of Probability Update Policy

The proposed PLTS algorithm employs a probability update mechanism to guide the search during the perturbation procedure (Section 4.5). In this subsection, we provide an analysis of the reward factor β and the penalization factor γ involved in the probability update mechanism. Specifically, we study β and γ independently by fixing one of these values to 0.5 and varying another value from 0.1 to 0.9 with a step size of 0.1. This experiment is based on the same eight instances introduced in Section 6.1. We ran PLTS 10 times with the same cut-off time as in Section 5.2. Figure 3 presents the average values of f_{best} achieved by our algorithm with β and γ on the eight instances tested.

From Figure 3, we can observe that PLTS shows better performance with the parameter settings of $\beta = 0.5$ and $\gamma = 0.5$, which are the default settings of our algorithm. In fact, it is reasonable to give each set a half chance of being selected, since we do not have any prior information of each set. And the

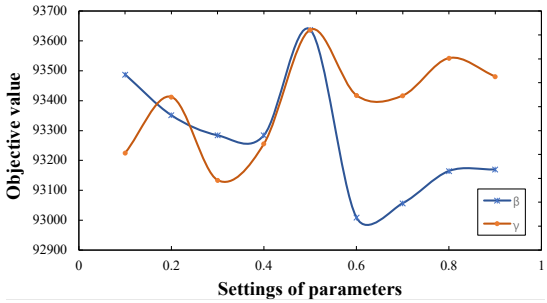

Fig. 3. Average of the f_{best} values on the eight instances obtained by executing PLTS with different values of β and γ

probability of each set will be updated by the Equation 7 and 8 as the search proceeds. In addition, the small *p-values* (4.42e-4 for β and 7.95e-4 for γ) from the Friedman test indicate that the differences from alternative parameter values are statistically significant.

6.3. Ablation Study on Perturbation Policy

We further analyze the main ingredients of the PLTS algorithm, the probability learning based perturbation. In Section 4.5, we present two strategies to escape from the local optimal solution, random perturbation and probability perturbation. Here we compare the two perturbation strategies, which allows us to better understand the behavior of the PLTS algorithm and shed light on its inner functioning.

To verify the effectiveness of the probability learning based perturbation used in the PLTS algorithm, we made a comparison between the probability perturbation and the random perturbation, in which we removed the probability learning mechanism from the PLTS algorithm. Denote the modified algorithm using the random perturbation as PLTS_0.

The investigation was conducted on the same set of instances we generated and each algorithm was run 30 times to solve each instance. The comparative results between PLTS and PLTS_0 are summarized in Table 6. For each instance, we report the best solution (f_{best}) and average solution (f_{avg}) of each algorithm, and better results (with a larger f_{best} or f_{avg}) between the two are in bold. The *p-values* from the Wilcoxon signed rank test are reported in the last row.

As shown in Table 6, the perturbation strategy exhibits a significant impact on the performance of our PLTS algorithm. PLTS improves on the best-known results for 10 out of 30 instances compared to the random perturbation. There are 19 instances where the two methods yield the same results, and only in

Table 6: Comparison of PLTS with $PLTS_0$ on the three sets of the BMCP instances.

Instance	PLTS		$PLTS_0$	
	f_{best}	f_{avg}	f_{best}	f_{avg}
bmcp_585_600_0.05_2000	71102	**71065.17**	71102	71056.97
bmcp_585_600_0.075_1500	70677	70677.00	70677	70677.00
bmcp_685_700_0.05_2000	81227	**80585.73**	81227	80578.67
bmcp_685_700_0.075_1500	82955	**82951.40**	82955	82947.80
bmcp_785_800_0.05_2000	**92608**	**92587.60**	92599	92492.20
bmcp_785_800_0.075_1500	94245	**94245.00**	94245	94244.60
bmcp_885_900_0.05_2000	**102162**	**101331.53**	101834	101259.73
bmcp_885_900_0.075_1500	106577	105942.43	**106723**	**106056.07**
bmcp_985_1000_0.05_2000	**109567**	**109408.77**	109470	109042.17
bmcp_985_1000_0.075_1500	**114969**	**113838.07**	114567	113583.03
bmcp_600_600_0.05_2000	68738	68472.00	68738	**68488.60**
bmcp_600_600_0.075_1500	71746	71746.00	71746	71746.00
bmcp_700_700_0.05_2000	**78028**	77859.27	77910	**77880.17**
bmcp_700_700_0.075_1500	84576	**84375.70**	84576	84257.03
bmcp_800_800_0.05_2000	91795	**91576.27**	91795	91392.07
bmcp_800_800_0.075_1500	95533	**95509.60**	95533	95500.73
bmcp_900_900_0.05_2000	101265	**101231.17**	101265	101165.93
bmcp_900_900_0.075_1500	104521	104521.00	104521	104521.00
bmcp_1000_1000_0.05_2000	**112802**	**111897.07**	112597	111560.43
bmcp_1000_1000_0.075_1500	**120246**	**118467.87**	119533	118453.60
bmcp_600_585_0.05_2000	67636	**67460.80**	67636	67373.20
bmcp_600_585_0.075_1500	70588	**70406.63**	70588	70357.63
bmcp_700_685_0.05_2000	78054	**78037.00**	78054	77992.67
bmcp_700_685_0.075_1500	78869	78869.00	78869	78869.00
bmcp_800_785_0.05_2000	**89138**	88581.20	89084	**88611.63**
bmcp_800_785_0.075_1500	91021	**91010.20**	91021	91006.70
bmcp_900_885_0.05_2000	98840	98718.00	98840	**98747.57**
bmcp_900_885_0.075_1500	**105141**	**104397.93**	105076	104253.67
bmcp_1000_985_0.05_2000	**111859**	**111228.80**	111802	111154.23
bmcp_1000_985_0.075_1500	112250	**112125.87**	112250	111885.13
p-value	-	-	2.08e-5	2.62e-3

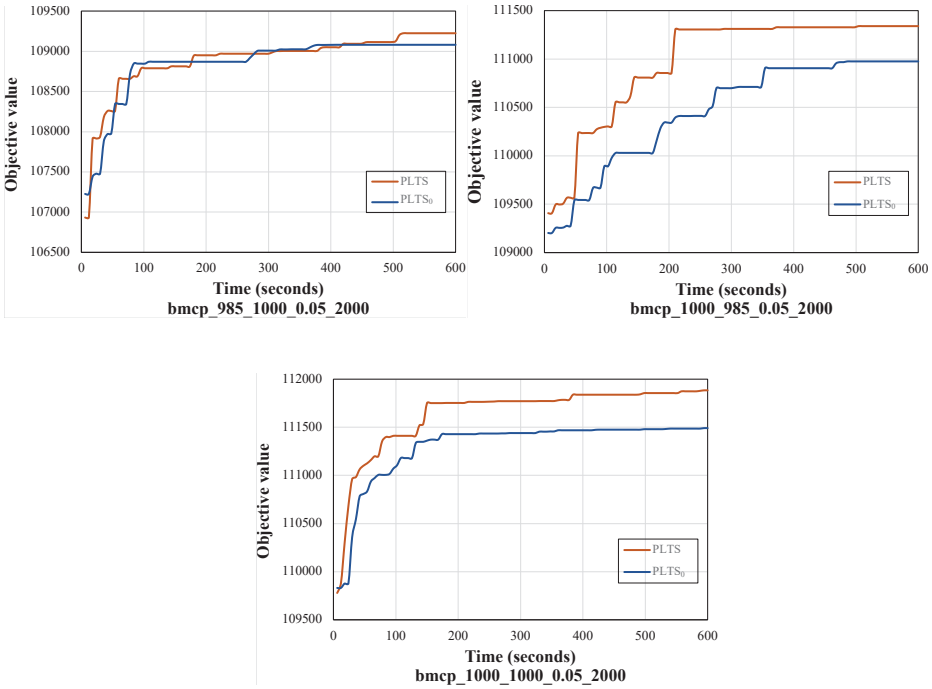

Fig. 4. Convergence graphs of PLTS and PLTS_0 for solving three BMCP instances.

one case the random perturbation achieves a better result. As for the average solution, there are 21 out of 30 instances that PLTS yields better results. Only on 5 instances PLTS_0 yields better solutions and on 4 instances the two methods yield the same results. Moreover, the *p-values* (< 0.05) from the Wilcoxon signed rank test disclose that the difference between PLTS and PLTS_0 is significant. The study indicates that the probability perturbation strategy with probability vector is significantly better than the simple random perturbation based PLTS_0 algorithm.

To further study the behaviors of the two perturbation strategies, we carried out an additional experiment to observe the convergence graphs of the algorithms. This experiment is based on three representative BMCP instances of different sets, i.e., 985_1000_0.05_2000, 1000_985_0.05_2000, 1000_1000_0.05_2000. We ran PLTS and PLTS_0 30 times to solve each of the selected instances with the same cut-off time (600 seconds). The best objective values during the search were recorded. Then we obtain the convergence graphs shown in Figure 4. The $X - axis$ in each sub-figure gives the running time of the algorithm and the $Y - axis$ indicates the best objective values found by PLTS and PLTS_0.

From Figure 4, we can observe that the proposed PLTS algorithm is able to discover better solutions with the increase of time. We can also observe that PLTS always obtains better best objective values than $PLTS_0$ when the cut-off time is reached, indicating that PLTS outperforms $PLTS_0$. The competitiveness of the PLTS algorithm becomes even more evident when we consider the convergence graphs of 1000_985_0.05_2000, 1000_1000_0.05_2000. For these two instances, the curves of PLTS almost strictly run above the curves of $PLTS_0$, indicating that our PLTS algorithm can improve the solution quality quickly and achieve better results. For the instance 985_1000_0.05_2000, although $PLTS_0$ shows similar performance to PLTS, our PLTS algorithm has better performance with the increase of time. This experiment again confirms the effectiveness of the proposed PLTS algorithm.

7. Conclusions

In this paper, we present a probability learning based tabu search (PLTS) algorithm to efficiently solve the Budgeted Maximum Coverage Problem (BMCP), which is a generalization of the NP-hard set cover problem as well as the NP-hard 0-1 knapsack problem, and a counterpart problem of the NP-hard Set-Union Knapsack Problem (SUKP). The proposed PLTS algorithm combines probability learning techniques and a tabu search procedure within the iterated local search framework. Probability learning is used to maintain and update a probability vector, with each entry specifying the probability that the corresponding set is selected. At each iteration, the PLTS algorithm applies a tabu search procedure to improve the solution until a local optimum is reached, then the PLTS algorithm adopts a perturbation phase based on the probability vector to escape from the local trap.

We generated three sets with a total of 30 instances with varied characteristics and compared the results of our algorithm with the typical approximation algorithm and the genetic algorithm. We also employed the general CPLEX solver for solving the BMCP instances for the first time. For all the generated instances, the best results obtained by the PLTS algorithm were better than the results obtained by the approximation algorithm, the genetic algorithm and the lower bounds (LB) of the CPLEX solver. In the section of analysis, we carried out a parameter analysis and discussed the effects of tabu search depth, reward factor and penalization factor on the performance of the PLTS algorithm. We also showed an ablation study on the probability learning based perturbation and made a comparison between probability perturbation and random pertur-

bation, demonstrating that the probability perturbation is much more effective than the random perturbation for solving the BMCP.

The way of using probability learning method in the proposed algorithm is innovative. Still there is room for further improvement. In future work, it is worth testing other local search algorithms or applying other heuristics for solving the BMCP. Also, our probability learning based tabu search approach could be applied to other combinatorial optimization problems, such as various variants of the knapsack problems and the set cover problems.

Acknowledgments

We are grateful to the reviewers for their useful comments and suggestions, which helped us to significantly improve the paper. The work is partially supported by the "Sino-French CAI Yuanpei Program" (Grant No. 41342NC). Support from the China Scholarship Council (Grant No. 201706290016) for Zequn WEI is also acknowledged.

References

References

Arulselvan, A., 2014. A note on the set union knapsack problem. Discrete Applied Mathematics 169, 214–218. URL: https://doi.org/10.1016/j.dam.2013.12.015.

Balas, E., Padberg, M.W., 1972. On the set-covering problem. Operations Research 20, 1152–1161. URL: https://doi.org/10.1287/opre.20.6.1152.

Benlic, U., Epitropakis, M.G., Burke, E.K., 2017. A hybrid breakout local search and reinforcement learning approach to the vertex separator problem. European Journal of Operational Research 261, 803–818. URL: https://doi.org/10.1016/j.ejor.2017.01.023.

Boyan, J.A., Moore, A.W., 2000. Learning evaluation functions to improve optimization by local search. Journal of Machine Learning Research 1, 77–112. URL: http://jmlr.org/papers/v1/boyan00a.html.

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

Cohen, R., Katzir, L., 2008. The generalized maximum coverage problem. Information Processing Letters 108, 15–22. URL: https://doi.org/10.1016/j.ipl.2008.03.017.

Curtis, D.E., Pemmaraju, S.V., Polgreen, P., 2010. Budgeted maximum coverage with overlapping costs: monitoring the emerging infections network, in: 2010 Proceedings of the Twelfth Workshop on Algorithm Engineering and Experiments (ALENEX), Austin, Texas, USA, pp. 112–123.

Glover, F.W., Laguna, M., 1997. Tabu Search. Kluwer.

Goldschmidt, O., Nehme, D., Yu, G., 1994. Note: On the set-union knapsack problem. Naval Research Logistics 41, 833–842. URL: https://doi.org/10.1002/1520-6750(199410)41:6<833::AID-NAV3220410611>3.0.CO;2-Q.

He, Y., Xie, H., Wong, T., Wang, X., 2018. A novel binary artificial bee colony algorithm for the set-union knapsack problem. Future Generation Computer Systems 78, 77–86. URL: https://doi.org/10.1016/j.future.2017.05.044.

Hochba, D.S., 1997. Approximation algorithms for np-hard problems. ACM Sigact News 28, 40–52. URL: https://doi.org/10.1145/261342.571216.

Kar, B., Wu, E.H., Lin, Y., 2016. The budgeted maximum coverage problem in partially deployed software defined networks. IEEE Transactions on Network and Service Management 13, 394–406. URL: https://doi.org/10.1109/TNSM.2016.2598549.

Kellerer, H., Pferschy, U., Pisinger, D., 2004. Knapsack problems. Springer.

Khuller, S., Moss, A., Naor, J., 1999. The budgeted maximum coverage problem. Information Processing Letters 70, 39–45. URL: https://doi.org/10.1016/S0020-0190(99)00031-9.

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

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. URL: https://doi.org/10.1016/j.eswa.2019.06.007.

Narendra, K.S., Thathachar, M.A.L., 1989. Learning Automata - An Introduction. Prentice Hall.

Nemhauser, G.L., Wolsey, L.A., Fisher, M.L., 1978. An analysis of approximations for maximizing submodular set functions - I. Mathematical Programming 14, 265–294. URL: https://doi.org/10.1007/BF01588971.

Piva, B., 2019. Approximations for restrictions of the budgeted and generalized maximum coverage problems, in: Proceedings of the tenth Latin and American Algorithms, Graphs and Optimization Symposium, Belo Horizonte, Brazil, pp. 667–676.

van Heuven van Staereling, I., de Keijzer, B., Schäfer, G., 2016. The ground-set-cost budgeted maximum coverage problem, in: 41st International Symposium on Mathematical Foundations of Computer Science, MFCS 2016, Kraków, Poland, pp. 50:1–50:13.

Suh, K., Guo, Y., Kurose, J.F., Towsley, D.F., 2006. Locating network monitors: Complexity, heuristics, and coverage. Computer Communications 29, 1564–1577. URL: https://doi.org/10.1016/j.comcom.2005.07.009.

Taylor, R., 2016. Approximations of the densest k-subhypergraph and set union knapsack problems. CoRR abs/1610.04935. URL: http://arxiv.org/abs/1610.04935.

Wang, X., Tang, L., 2017. A machine-learning based memetic algorithm for the multi-objective permutation flowshop scheduling problem. Computers & Operations Research 79, 60–77. URL: https://doi.org/10.1016/j.cor.2016.10.003.

Wang, Y., Pan, S., Li, C., Yin, M., 2020. A local search algorithm with reinforcement learning based repair procedure for minimum weight independent dominating set. Information Sciences 512, 533–548. URL: https://doi.org/10.1016/j.ins.2019.09.059.

Wei, Z., Hao, J.K., 2019. Iterated two-phase local search for the set-union knapsack problem. Future Generation Computer Systems 101, 1005–1017. URL: https://doi.org/10.1016/j.future.2019.07.062.

Wei, Z., Hao, J.K., 2020. Kernel based tabu search for the set-union knapsack problem. Expert Systems with Applications 165, 113802. URL: https://doi.org/10.1016/j.eswa.2020.113802.

Wei, Z., Hao, J.K., 2021. Multistart solution-based tabu search for the set-union knapsack problem. Applied Soft Computing , 107260doi:https://doi.org/10.1016/j.asoc.2021.107260.

Wu, C., He, Y., 2020. Solving the set-union knapsack problem by a novel hybrid jaya algorithm. Soft Computing 24, 1883–1902. URL: https://doi.org/10.1007/s00500-019-04021-3.

Zhou, Y., Duval, B., Hao, J.K., 2018. Improving probability learning based local search for graph coloring. Appllied Soft Computing 65, 542–553. URL: https://doi.org/10.1016/j.asoc.2018.01.027.

Zhou, Y., Hao, J.K., Duval, B., 2016. Reinforcement learning based local search for grouping problems: A case study on graph coloring. Expert Systems with Applications 64, 412–422. URL: https://doi.org/10.1016/j.eswa.2016.07.047.