Efficient and Effective Local Search for the Set-Union Knapsack Problem and Budgeted Maximum Coverage Problem

Wenli Zhu\textsuperscript{a}, Liangqing Luo\textsuperscript{b}

\textsuperscript{a}School of Statistics, Jiangxi University of Finance and Economics, Nanchang Jiangxi 330013, China; Email: zzwenu@163.com
\textsuperscript{b}School of Statistics, Jiangxi University of Finance and Economics, Nanchang Jiangxi 330013, China; Corresponding author; Email: llq429@163.com

\textbf{ARTICLE INFO}

\textbf{Keywords:}
Set-union knapsack problem
Budgeted maximum coverage problem
Local search
Tabu search

\textbf{ABSTRACT}

The Set-Union Knapsack Problem (SUKP) (Goldschmidt et al., 1994) and Budgeted Maximum Coverage Problem (BMCP) (Khuller et al., 1999) are two closely related NP-hard combinatorial optimization problems. Let \( I = \{i_1, \ldots, i_m\} \) be a set of \( m \) items where each item \( i_j, j \in \{1, \ldots, m\} \) has a nonnegative value \( v_j \). \( E = \{e_1, \ldots, e_n\} \) be a set of \( n \) elements where each element \( e_k, k \in \{1, \ldots, n\} \) has a nonnegative weight \( w_k \), and \( C \) is the capacity of a given knapsack in the SUKP (or the budget in the BMCP). The items and elements are associated by a relation matrix \( R \in \{0, 1\}^{m \times n} \), where \( R_{jk} = 1 \) indicates that \( e_k \) is covered by \( i_j \), otherwise \( R_{jk} = 0 \). The SUKP aims to find a subset \( S \) of \( I \) that maximizes the total value of the items in \( S \), at the same time the total weight of the elements covered by the items in \( S \) does not exceed the capacity \( C \). The SUKP can be stated formally as follows.

\begin{align*}
\text{Maximize} & \quad f(S) = \sum_{j \in \{j \mid i_j \in S\}} v_j, \quad (1) \\
\text{Subject to} & \quad W(S) = \sum_{k \in \{k \mid R_{jk} = 1, j \in S\}} w_k \leq C. \quad (2)
\end{align*}

For the BMCP, the goal is to find a subset \( S \) of \( I \) that maximizes the total weight of the elements covered by the items in \( S \), while the total value of the items in \( S \) does not exceed the capacity (budget) \( C \). The BMCP can be stated formally as follows.

\begin{align*}
\text{Maximize} & \quad W(S), \quad (3) \\
\text{Subject to} & \quad f(S) \leq C. \quad (4)
\end{align*}

Obviously, the SUKP and BMCP can be transferred to each other by swapping the optimization objective and constraint objective. Both the SUKP and BMCP are computationally challenging and have many real-world applications, such as flexible manufacturing (Goldschmidt et al., 1994), financial decision making (Khuller et al., 1999; Kellner et al., 2004), data compression (Yang et al., 2016), software defined network optimization (Kar et al., 2016), project investment (Wei and Hao, 2019), etc.

Exact and approximation algorithms are two kinds of methods for the SUKP and BMCP. For example, exact algorithms for the SUKP based on dynamic programming (Arulselvan, 2014) and linear integer programming (Wei and Hao, 2019), and greedy approximation algorithms for the SUKP (Taylor, 2016) and BMCP (Khuller et al., 1999). These algorithms can all theoretically guarantee the quality of their solutions, but the exact algorithms are hard to scale to large instances, and the approximation algorithms are hard to yield high-quality results.

Heuristic algorithms such as population-based algorithms and local search algorithms are more practical than exact and approximation algorithms. Population-based methods usually use bio-inspired metaheuristic operations among the population, so as to find excellent individuals. He et al. (He et al., 2018) first proposed a binary artificial bee colony algorithm for the SUKP. Later on, a swarm intelligence-based algorithm (Özsoydan and Baykasoglu, 2019) was proposed to solve the SUKP. There are also some hybrid algorithms for the SUKP that combine population-based methods with local search to improve the performance. For example, Lin et al. (Lin et al., 2019) combined binary particle swarm optimization with tabu search, Dahmani et al. (Dahmani et al., 2020) combined swarm optimization with local search operators, Wu and He (Wu and He, 2020) proposed a hybrid Jaya algorithm for the SUKP.
Population-based methods have attracted lots of attention. However, these methods are more complex to design than local search algorithms, and the performance of existing population-based methods is not as good as the state-of-the-art local search methods for the SUKP.

For the local search methods, the I2PLS algorithm (Wei and Hao, 2019) is the first local search method for the SUKP, which is based on tabu search. Later on, some better tabu search methods were proposed (Wei and Hao, 2021a,b). Wei and Hao (Wei and Hao, 2021b) tested these tabu search algorithms on SUKP instances with no more than 1,000 items and elements. Recently, Zhou et al. (Zhou et al., 2021) proposed an efficient local search algorithm called ATS-DLA for the SUKP, and tested the algorithm on SUKP instances with up to 5,000 items and elements. For the BMCP, Li et al. (Li et al., 2021) proposed the first local search method. Zhou et al. (Zhou et al., 2022) proposed a local search algorithm based on a partial depth-first search tree.

Local search algorithms have obtained excellent results for solving the SUKP and BMCP. However, there are still some disadvantages of the existing local search algorithms for these two problems. The I2PLS (Wei and Hao, 2019), KBTS (Wei and Hao, 2021a), MSBTS (Wei and Hao, 2021b), and PLTS (Li et al., 2021) algorithms all tend to find the best neighbor solution of the current solution in each iteration. However, their search neighborhood contains lots of low-quality moves, which may reduce the algorithm efficiency. The search region of the ATS-DLA algorithm (Zhou et al., 2021) is small, which may make the algorithm hard to escape from some local optima. The VDLS algorithm (Zhou et al., 2022) has a wide and deep search region. However, VDLS does not allow the current solution worse than the previous one, which may restrict the algorithm’s search ability. In summary, these local search methods cannot trade-off the efficiency and search region well.

To handle this issue, we propose an efficient and effective local search algorithm, called E2LS, for both the SUKP and BMCP. To the best of our knowledge, this is the first time that an algorithm has been proposed to solve the SUKP and BMCP simultaneously. E2LS uses a random greedy algorithm to generate the initial solution, and an effective local search method to explore the solution space. E2LS restricts the items that can be removed from or added into the current solution to improve the search efficiency. In this way, E2LS can refine the search region by abandoning low-quality candidates. The local search operator in E2LS then traverses the refined search region to find high-quality moves. Thus E2LS can explore the solution space widely and quickly, so as to find high-quality solutions. Moreover, the tabu search method in (Wei and Hao, 2021b) is used in E2LS to prevent the algorithm from getting stuck in local optima. Indeed, as we have shown in this work, our proposed E2LS algorithm significantly outperforms the state-of-the-art heuristic algorithms for both the SUKP and BMCP.

The main contributions of this work are as follows.

- E2LS trade-offs the search region and search efficiency well. E2LS restricts the items that can be removed from or added into the current solution, so as to abandon low-quality moves and refine the search region. The proposed operator ADD\textsuperscript{*} can traverse the refined search region, so as to explore the solution space widely and efficiently.
- The mechanism that trade-offs the search region and efficiency, as well as the method of traversing the refined search region, could be applied to other combinatorial optimization problems.
- Extensive experiments demonstrate that E2LS significantly outperforms the state-of-the-art algorithms for both the SUKP and BMCP. In particular, E2LS provides four new best-known solutions for the SUKP, and 27 new best-known solutions for the BMCP.

2. The Proposed E2LS Algorithm

This section introduces the proposed E2LS algorithm. We first present the components of E2LS, including the random greedy initialization method and the searching methods, then present the main process of E2LS. Finally, we discuss the advantages of E2LS over other state-of-the-art local search algorithms for the SUKP and BMCP. Note that the proposed E2LS algorithm can be used to solve both the SUKP and BMCP. This section mainly introduces its SUKP version. The BMCP version can be obtained simply by swapping the optimization objective and constraint objective of the SUKP version.

Before introducing our method, we first present several essential definitions used in the E2LS algorithm.

Definition 1. (Additional Weight) Note that $W(S)$ (see Eq. 2) is the total weight of the elements covered by the items in $S$. Let $AW(S,i_j)$ to be the additional weight of an item $i_j$ to a solution $S$. If $i_j \notin S$, the additional weight $AW(S,i_j) = W(S \cup \{i_j\}) - W(S)$ represents the increase of the total weight of the covered elements caused by adding $i_j$ to $S$. Otherwise, $AW(S,i_j) = W(S) - W(S \setminus \{i_j\})$ represents the decrease of the total weight of the covered elements caused by removing $i_j$ from $S$.

Definition 2. (Value-weight Ratio) The value-weight ratio of an item $i_j$ to a solution $S$ is defined as $vw(S,i_j) = \frac{v_j}{AW(S,i_j)}$, which is the ratio of the value of $i_j$ to the addition weight of $i_j$ to $S$. Obviously, an item with a larger value-weight ratio to a solution $S$ is a better candidate item of $S$ (Khuller et al., 1999; Zhou et al., 2021, 2022).

2.1. Random Greedy Initialization Method

We propose a simple construction method to generate the initial solution for E2LS. The procedure of the initialization method is shown in Algorithm 1.

The algorithm starts with an empty solution $S$, and repeats to add items into $S$ until no more items can be added
into \( S \) (line 8). In each loop, each item \( i_j \notin S \) with \( AW(S, i_j) = 0 \) will be added into \( S \) (line 6). Such an operation can increase \( f(S) \) (see Eq. 1) without increasing \( W(S) \). When there is at least one item that adding into \( S \) will result in a feasible solution, i.e., the feasible candidates \( FC \neq \emptyset \), the algorithm applies the probabilistic sampling strategy (Cai, 2015; Zheng et al., 2021) to select the item to be added. Specifically, the algorithm first random samples \( t \) items with replacement in \( FC \) (lines 11-12), then selects to add the item with the maximum value-weight ratio (lines 13-15).

We set the parameter \( t \) to be \( \sqrt{\max\{m,n\}} \) as the algorithm MSBTS (Wei and Hao, 2021b) does. This setting can help the E2LS algorithm yield high-quality and diverse initial solutions. In particular, we analyze the influence of \( t \) on the performance of E2LS in experiments. The results show that E2LS is very robust and not sensitive to the parameter \( t \). Even with a random initialization method (i.e., \( t = 1 \)), E2LS also has excellent performance.

2.2. Searching Methods in E2LS

The local search method in E2LS is the main improvement of E2LS to other local search algorithms for the SUKP and BMCP. In E2LS, we propose an efficient and effective local search operator to explore the solution space. We also apply the solution-based tabu search method in (Wei and Hao, 2021b) to avoid getting stuck in local optima. This subsection first describes how to represent the tabu list, then introduces the local search process.

2.2.1. Tabu List Representation

The tabu search method in (Wei and Hao, 2021b) uses three hash vectors \( H_1, H_2, H_3 \) to represent the tabu list. The length of each vector is set to \( L \) (\( L = 10^8 \) by default). The three hash vectors are initialized to 0, indicating that no solution is prohibited by the tabu list. Each solution \( S \) is corresponding to three hash values \( h_1(S), h_2(S), h_3(S) \). A solution \( S \) is prohibited (i.e., in the tabu list) if \( H_i[h_1(S)] \land H_2[h_2(S)] \land H_3[h_3(S)] = 1 \). See the details for calculating the hash values of a solution below.

For an instance with \( m \) items, a weight matrix \( W \in \mathbb{N}^{[1\times m]} \) is calculated as follows. First, let \( W_{ij} = [j^2], i = (1,2,3), j = (1,...,m) \), where \( \gamma_1, \gamma_2, \gamma_3 \) are set to 1.2, 1.6, 2.0, respectively. Then, random shuffle each of the three rows of \( W \). Given a solution \( S \) that can be represented by \((y_1,...,y_m)\) where \( y_j = 1 \) if \( i_j \in S \), and \( y_j = 0 \) otherwise. The hash values \( h_i(S) \), \( i = (1,2,3) \) can be calculated by \( h_i(S) = (\sum_{j=1}^{m} [\gamma_i y_j]) \mod L \).

2.2.2. Local Search Process

The procedure of the local search method in E2LS is shown in Algorithm 2. The search operator first removes an item from the input solution \( S \) (line 6), and then uses the proposed operator \( ADD^* \) (Algorithm 3) to add items into the resulting solution \( S' \) (line 8). The best solution that is not in the tabu list found during the search process is represented by \( S_b \) (line 9).

As shown in Algorithm 3, the function \( ADD^* \) tries to add items into the input solution \( S \) recursively (line 18) until no more items can be added into \( S \) (line 11). The best solution that is not in the tabu list found during the process is represented by \( S_b \) (lines 8 and 17). The algorithm first calculates the set of the feasible candidate items \( FC \) of \( S \) (lines 1-11). During the process, each item \( i_j \notin S \) with \( AW(S, i_j) = 0 \) will be added into \( S \) if the resulting solution is not in the tabu list (lines 5-7). After that, the algorithm traverses part of the set \( FC \) to add an item \( i \) into \( S \) (lines 13-14), and then calls the function \( ADD^* \) with the input solution \( S \cup \{i\} \) to continue the search (line 18). During the process, the solutions in the tabu list are prohibited (line 15).

---

**Algorithm 1: Random_Greedy(t)**

**Input:** Sampling times \( t \)

**Output:** Solution \( S \)

1. Initialize \( S \leftarrow \emptyset \);
2. while TRUE do
   1. Initialize feasible candidates \( FC \leftarrow \emptyset \);
   2. for \( j \leftarrow 1 : n \) do
      1. if \( i_j \notin S \) then continue;
      2. if \( AW(S, i_j) = 0 \) then \( S \leftarrow S \cup \{i_j\} \);
      3. if \( AW(S, i_j) + W(S) \leq C \) then \( FC \leftarrow FC \cup \{i_j\} \);
   3. if \( FC = \emptyset \) then break;
   4. else
      1. \( M \leftarrow 0 \);
      2. for \( j \leftarrow 1 : t \) do
         1. \( i_r \leftarrow \) a random item in \( FC \);
         2. if \( R_{vw}(S, i_r) > M \) then \( M \leftarrow R_{vw}(S, i_r), i_b \leftarrow i_r \);
      3. \( S \leftarrow S \cup \{i_b\} \);
   5. return \( S \);

**Algorithm 2: Local_Search(S, r_{num}, a_{num})**

**Input:** Input solution \( S \), maximum size of the candidate set of the items to be removed \( r_{num} \), maximum size of the candidate set of the items to be added \( a_{num} \)

**Output:** Output solution \( S' \)

1. \( S_b \leftarrow \emptyset \);
2. \( U \leftarrow \) a set of items in \( S \);
3. Sort the items in \( U \) in ascending order of their value-weight ratios to \( S \);
4. for \( j \leftarrow 1 : \min(r_{num}, |U|) \) do
   1. \( i \leftarrow j \)-th item in \( U \);
   2. \( S' \leftarrow S \setminus \{i\} \);
   3. if \( S' \) is in the tabu list then
      1. \( S' \leftarrow ADD^*(S', \emptyset, a_{num}); \)
      2. if \( f(S') > f(S_b) \) then \( S_b \leftarrow S' \);
5. return \( S_b \).
The proposed function ADD$^*$ is powerful and actually has a wide search region. The reasons are as follows. Suppose we set $a_{num} = m$, ADD$^*$ can traverse all the combinations of the candidate items by the recursive process, i.e., ADD$^*$ can find the optimal solution corresponding to its input partial solution when the items in its input solution are fixed. In particular, the function ADD$^*$ can be regarded as an exact solver for an instance with $m$ items if we call ADD$^*(\emptyset, \emptyset, m)$. However, an exhaustive search must be inefficient.

In order to improve the efficiency of the local search process, the low-quality moves (candidate items) should not be considered during the search process. To yield high-quality results, the items with large value-weight ratios to the current solution should be added into (or be kept in) it, and the items with small value-weight ratios should be removed from (or not be added into) it. Therefore, in Algorithm 2, the items that can be removed from $S$ are restricted by a parameter $r_{num}$. That is, only the top $\min(r_{num}, |S|)$ items with minimum value-weight ratios to $S$ can be removed from $S$ (lines 3-4). In Algorithm 3, only the top $\min(a_{num}, |FC|)$ items with maximum value-weight ratios to $S$ can be added into $S$ (lines 12-13). By applying this strategy, the search region can be refined significantly, i.e., the number of items considered by ADD$^*$ is reduced from $|FC|$ to $\min(a_{num}, |FC|)$, and the efficiency can be improved greatly. Also, the operator $ADD^*$ does not traverse all the moves, but only traverses the refined search region.

Moreover, our proposed ADD$^*$ operator can add multiple items into the current solution $S$, which can not be regarded as the same as the combination of multiple continuous ADD operators in (Wei and Hao, 2019, 2021a) or flip operators (on items not in $S$) in (Wei and Hao, 2021b; Li et al., 2021). For example, the ADD$^*$ operator adds items $i_j$ into $S$, while the best neighbor solution of $S$ find by the commonly used ADD operator might chooses to add item $i_k$. In summary, ADD$^*$ effective because it can traverse various combinations of the items in the refined search region that can be added into the current solution.

### 2.2.3. Main Framework of E2LS

The main process of E2LS is shown in Algorithm 4. E2LS first initializes the three hash vectors $H_1$, $H_2$, $H_3$ to 0 (line 2), and calls the Random_Greedy function (Algorithm 1) to generate the initial solution $S$ (line 3). Then, E2LS calls the Local_Search function (Algorithm 2) to explore the solution space until the cut-off time is reached (lines 5-12). During the iterations, the Random_Greedy function will be called again when the Local_Search function with the input solution $S$ can not find a solution that is not in the tabu list (line 7-8), i.e., the output solution of the Local_Search function is $\emptyset$. The solution $S$ obtained in each iteration will be added into the tabu list by setting $H_1[h(S)] \land H_2[h(S)] \land H_3[h(S)]$ to 1 (line 11). The best solution found so far is represented by $S_b$ (line 12).

### 2.2.4. Advantages of E2LS

This subsection discusses the main advantages of our proposed E2LS over the state-of-the-art local search algorithms for the SUKP and BMCP.

The first category of local search methods includes the

---

**Algorithm 3: ADD$^*(S, S_b, a_{num})$**

**Input:** Input solution $S$, best solution found in this step $S_b$, maximum size of the candidate set of the items to be added $a_{num}$

**Output:** Output solution $S$

1. while TRUE do
   2.  for $j \leftarrow 1 : m$ do
      3.     if $i_j \in S$ then continue;
      4.     if $AW(S, i_j) = 0$ then
      5.         $S \leftarrow S \cup \{i_j\}$;
      6.         if $f(S) > f(S_b)$ then $S_b \leftarrow S$;
      7.     else if $AW(S, i_j) + W(S) \leq C$ then
      8.         $FC \leftarrow FC \cup \{i_j\}$;
      9.     end if
   10.    end if
   11.   end for
   12.   Sort the items in $FC$ in descending order of their value-weight ratios to $S$;
   13.   for $j \leftarrow 1 : \min(a_{num}, |FC|)$ do
      14.     $i \leftarrow j$-th item in $FC$;
      15.     if $S \cup \{i\}$ is in the tabu list then continue;
      16.     $S' \leftarrow S \cup \{i\}$;
      17.     if $f(S') > f(S_b)$ then $S_b \leftarrow S'$;
      18.     $S' \leftarrow ADD^*(S', S_b, a_{num})$;
   19.   end for
   20. return $S_b$;

**Algorithm 4: E2LS(I, $T_{max}$, $t$, $r_{num}$, $a_{num}$)**

**Input:** Instance $I$, cut-off time $T_{max}$, sampling times $t$, maximum size of the candidate set of the items to be removed $r_{num}$, maximum size of the candidate set of the items to be added $a_{num}$

**Output:** Output solution $S$

1. Read instance $I$;
2. Initialize $H_1$, $H_2$, $H_3$ to 0;
3. $S \leftarrow$ Random_Greedy($t$);
4. Initialize $S_b \leftarrow \emptyset$;
5. while the cut-off time $T_{max}$ is not reached do
   6.     $S' \leftarrow$ Local_Search($S$, $r_{num}$, $a_{num}$);
   7.     if $S' = \emptyset$ then
   8.         $S \leftarrow$ Random_Greedy($t$);
   9.         continue;
   10.    $S \leftarrow S'$;
   11.    for $j \leftarrow 1 : 3$ do $H_i[h(S)] \leftarrow 1$;
   12.    if $f(S) > f(S_b)$ then $S_b \leftarrow S$;
   13. return $S_b$;
I2PLS (Wei and Hao, 2019), KBTS (Wei and Hao, 2021a), MSBTS (Wei and Hao, 2021b) algorithms for the SUKP, and the PLTS (Li et al., 2021) algorithm for the BMCP. These algorithms are all based on tabu search. Their common shortcoming is that their search neighborhoods contain lots of low-quality moves. While E2LS refines the search neighborhood according to the value-weight ratios of the items. Thus, E2LS shows much better performance and higher efficiency than these algorithms.

The second category is the ATS-DLA algorithm (Zhou et al., 2021), which also uses the tabu search method and is efficient for large scale SUKP instances. However, ATS-DLA can only remove or add one item in each iteration, which leads to a relatively small and overrefined search region, and may make it hard to escape from local optima in some cases. While E2LS can add multiple items in each iteration by traversing the refined search region, and considering various combinations of the items. Therefore, E2LS can explore the solution space wider and deeper than ATS-DLA, so as to find higher-quality solutions.

The third category is the VDLS algorithm (Zhou et al., 2022), which is not based on tabu search but a partial depth-first search method. VDLS has a large search region, and can yield significantly better solutions than the PLTS algorithm (Li et al., 2021). However, VDLS does not allow the solution to get worse during the search process, and the initial solution generated by VDLS is fixed. Such mechanisms limit the search ability of the algorithm. E2LS does not restrict that the current solution must be better than the previous one, and applies the tabu search method to avoid getting stuck in local optima. Moreover, the random greedy initialization method in E2LS can generate high-quality and diverse initial solutions for E2LS. Thus E2LS also shows significantly better performance than VDLS.

3. Computational Results

This section presents experimental results and analyses, before which we first introduce the benchmark instances used in the experiments and the experimental setup.

3.1. Benchmark Instances

We tested E2LS on a total of 78 public SUKP instances with at most 5,000 items or elements, and 90 public BMCP instances with no more than 5,200 items or elements. The 78 tested SUKP instances can be divided into three sets as follows.

- **Set I:** This set contains 30 instances with 85 to 500 items or elements. This set was proposed in (He et al., 2018) and widely used in (He et al., 2018; Ozsoydan and Baykasoglu, 2019; Lin et al., 2019; Wei and Hao, 2019; Dahmani et al., 2020; Wu and He, 2020; Wei and Hao, 2021a,b).

- **Set II:** This set contains 30 instances with 85 to 1,000 items or elements. This set was proposed in (Wei and Hao, 2021a) and used in (Wei and Hao, 2021a,b).

- **Set III:** This set contains 18 instances with 850 to 5,000 items or elements. This set was proposed in (Zhou et al., 2021).

Each instance in Sets I, II, and III is characterized by four parameters: the number of items \(m\), the number of elements \(n\), the density of the relation matrix \(a = (\sum_{j=1}^{m} \sum_{i=1}^{n} R_{ij})/(mn)\), and the ratio of knapsack capacity \(C\) to the total weight of the elements \(\beta = C/\sum_{i=1}^{n} w_{i}\). The name of a SUKP instance consists of these four parameters. For example, sukp_85_100_0.10_0.75 represents a SUKP instance with 85 items, 100 elements, \(a = 0.10\), and \(\beta = 0.75\).

The 90 tested BMCP instances can be divided into three sets as follows.

- **Set A:** This set contains 30 instances with 585 to 1,000 items or elements. This set was proposed in (Li et al., 2021) and used in (Li et al., 2021; Zhou et al., 2022).

- **Set B:** This set contains 30 instances with the number of items or elements ranging from 1,000 to 1,600. This set was proposed in (Zhou et al., 2022).

- **Set C:** This set contains 30 instances with the number of items or elements ranging from 4,000 to 5,200. This set was proposed in (Zhou et al., 2022).

Each instance in Set A is characterized by four parameters: the number of items \(m\), the number of elements \(n\), the knapsack capacity (budget) \(C\), and the density of the relation matrix \(a\). The instances in Sets B and C have more complex structures than those in Set A. Zhou et al. (Zhou et al., 2022) created these instances by first randomly grouping the items and elements, then deciding the connection of them according to a parameter \(\rho\) that represents the density of the relation matrix of each group. Each instance in Sets B and C is characterized by parameters \(m, n, C\) and \(\rho\).

3.2. Experimental Setup

We first introduce the baseline algorithms we selected. For the SUKP, we select some of the state-of-the-art heuristic algorithms, including ATS-DLA (Zhou et al., 2021), MS-BTS (Wei and Hao, 2021b), KBTS (Wei and Hao, 2021a), and I2PLS (Wei and Hao, 2019), as the baseline algorithms. For the BMCP, we select the heuristic algorithms PLTS (Li et al., 2021) and VDLS (Zhou et al., 2022) as the baseline algorithms. To our knowledge, they are also the only two heuristic algorithms for the BMCP.

The E2LS algorithm and the baseline algorithms were implemented in C++ and compiled by g++. All the experiments were performed on a server using an Intel® Xeon® E5-1603 v3 2.80GHz CPU, running Ubuntu 16.04 Linux operating system. The parameters in E2LS include the sampling times \(t\), and the maximum size of the candidate set of the items to be removed/added \(r_{num}/a_{num}\). We tuned these parameters according to our experience. The default value of \(t\) is set to \(\sqrt{\max\{m, n\}}\). For the parameters \(r_{num}\) and \(a_{num}\),
we set the default values of them to be different when solving the SUKP and BMCP, since the properties of these two problems are different. Specifically, for the SUKP, we set the default values as $r_{num} = 2$ and $a_{num} = 2$. For the BMCP, we set $r_{num} = 5$ and $a_{num} = 5$. The detailed reasons why their default values are different when solving the SUKP and BMCP, as well as the influence of these parameters on the performance of E2LS, are presented in Section 3.4.

We set the cut-off time for each algorithm to be 500 seconds for the instances in Set I (As Wei and Hao, 2021b) did, 1,000 seconds for the instances in Sets II and III as (Wei and Hao, 2021b; Zhou et al., 2021) did, 600 seconds for the instances in Set A as (Li et al., 2021) did, and 1,800 seconds for the instances in Sets B and C as (Zhou et al., 2022) did. Each instance was calculated 10 independent times for each algorithm.

### 3.3. Comparison with the baselines

The comparison results of E2LS with the baseline algorithms on the three sets of SUKP instances are presented in Tables 1, 2, and 3, respectively. The comparison results of E2LS with the baseline algorithms on the three sets of BMCP instances are presented in Tables 4, 5, and 6, respectively. In these tables, unique best results are appeared in bold, while equal best results are appeared in italic. Tables 1, 2, and 3 compare the best solution (objective value), average solution, standard deviations over 10 runs (S.D.), and the average run times in seconds (to obtain the best solution in each run) of each involved algorithm. Tables 4, 5, and 6 present the best solution and average solution of each involved algorithm, coupled with the standard deviations and average run times of E2LS. The results of PLTS and VDLS in Tables 4, 5, and 6 are from (Zhou et al., 2022) (they used the similar machine as ours, Intel® Xeon® E5-2650 v 3 2.30GHz).

Moreover, in order to show the advantage of E2LS over the baselines more clearly, we summarize the comparative results between E2LS and each baseline algorithm in Tables 7 and 8. Columns #Wins, #Ties, and #Losses indicate the number of instances for which E2LS obtains a better, equal, and worse result than the compared algorithm according to the best solution and average solution indicators.

As the results shown in Tables 1, 2, 3, and 7, E2LS does not lose to any baseline algorithm according to the best solution indicator. Actually, E2LS can obtain the best-known solution for all the 78 tested SUKP instances. E2LS also has good stability and robustness, since the average solutions obtained by E2LS are also excellent and the standard deviations of E2LS are very small. In particular, E2LS obtains three new best-known solutions for instances $sukp_{1000,850,0.15,0.85}$, $sukp_{3000,2850,0.10,0.75}$, and $sukp_{3000,2850,0.15,0.85}$.

Moreover, there are no baseline algorithms that can solve

---

1Note that there are two instances with the same name but the different structures in Set II and Set III.

2The result 9565 of instance $sukp_{1000,850,0.15,0.85}$ can also be obtained by MSBTS, but has not been reported in the literature. The result 9207 of instance $sukp_{5000,4850,0.10,0.75}$ has been reported in (Zhou et al., 2021).
Table 2: Comparison of E2LS with the baseline algorithms on 30 SUKP instances of Set II. Unique best results appear in bold. Equal best results appear in italic.

| Instance/Algorithm | Best | Average | S.D. | Time | EPS | Best | Average | S.D. | Time | Best | Average | S.D. | Time | Best | Average | S.D. | Time | Best | Average | S.D. | Time |
|--------------------|------|---------|------|------|-----|------|---------|------|------|------|---------|------|------|------|---------|------|------|------|---------|------|------|------|---------|------|------|
| sukp_600_500.txt | 9790 | 9791.50 | 12.64 | 315.89 | 8881 | 9804.60 | 0.00 | 181.513 | 9794 | 9796.00 | 0.00 | 154.247 | 9797 | 9798.40 | 0.00 | 144.406 | 9790 | 9790.00 | 0.00 | 154.247 | 9794 | 9796.00 | 0.00 | 154.247 |
| sukp_600_500.txt | 9137 | 9140.00 | 14.85 | 315.89 | 8881 | 9804.60 | 0.00 | 181.513 | 9137 | 9140.00 | 14.85 | 315.89 | 9140 | 9142.50 | 0.00 | 154.247 | 9137 | 9140.00 | 14.85 | 315.89 | 9140 | 9142.50 | 0.00 | 154.247 |
| sukp_600_500.txt | 9137 | 9140.00 | 14.85 | 315.89 | 8881 | 9804.60 | 0.00 | 181.513 | 9137 | 9140.00 | 14.85 | 315.89 | 9140 | 9142.50 | 0.00 | 154.247 | 9137 | 9140.00 | 14.85 | 315.89 | 9140 | 9142.50 | 0.00 | 154.247 |
| Average | 9274 | 9277.00 | 15.40 | 315.89 | 8881 | 9804.60 | 0.00 | 181.513 | 9274 | 9277.00 | 15.40 | 315.89 | 9277 | 9280.50 | 0.00 | 154.247 | 9274 | 9277.00 | 15.40 | 315.89 | 9277 | 9280.50 | 0.00 | 154.247 |

Table 3: Comparison of E2LS with the baseline algorithms on 18 SUKP instances of Set III. Unique best results appear in bold. Equal best results appear in italic.

| Instance/Algorithm | Best | Average | S.D. | Time | EPS | Best | Average | S.D. | Time | Best | Average | S.D. | Time | Best | Average | S.D. | Time | Best | Average | S.D. | Time |
|--------------------|------|---------|------|------|-----|------|---------|------|------|------|---------|------|------|------|---------|------|------|------|---------|------|------|------|---------|------|------|
| sukp_850_850.txt | 8485 | 8488.30 | 22.52 | 353.32 | 8657 | 8660.30 | 17.82 | 469.100 | 8673 | 8673.00 | 17.82 | 455.701 | 8679 | 8680.30 | 17.82 | 441.700 | 8685 | 8685.00 | 17.82 | 427.700 | 8691 | 8692.00 | 17.82 | 413.700 |
| sukp_1000_1000.txt | 10464 | 10471.30 | 44.88 | 840.53 | 10567 | 10570.30 | 22.52 | 884.100 | 10621 | 10621.00 | 22.52 | 816.100 | 10675 | 10678.30 | 22.52 | 748.100 | 10729 | 10732.00 | 22.52 | 680.100 | 10793 | 10796.00 | 22.52 | 612.100 |
| Average | 8583.50 | 8587.00 | 22.52 | 353.32 | 8657 | 8660.30 | 17.82 | 469.100 | 8673 | 8673.00 | 17.82 | 455.701 | 8679 | 8680.30 | 17.82 | 441.700 | 8685 | 8685.00 | 17.82 | 427.700 | 8691 | 8692.00 | 17.82 | 413.700 |

Zhu et al.: Preprint submitted to Elsevier
Efficient and Effective Local Search for the SUKP and BMCP
Table 4
Comparison of E2LS with the baseline algorithms on 30 BMCP instances of Set A. Unique/Equal best results appear in bold/italic.

| Instance/Algorithm | Best | Average | Best | Average | Best | Average | Best | Average | S.D. | Time |
|--------------------|------|---------|------|---------|------|---------|------|---------|------|------|
| bmcp_1100_1000_0.3_3000 | 142835 | 242486.70 | 24250 | 142426.00 | 24375 | 24367 | 0.00 | 1.117 | |
| bmcp_1100_1000_0.5_3000 | 11318 | 12509.00 | 11318 | 12509.00 | 11318 | 12509.00 | 0.00 | 0.519 | |
| bmcp_1100_1000_0.75_3000 | 112250 | 112160.20 | 112250 | 112160.20 | 112250 | 112160.20 | 0.00 | 0.452 | |
| bmcp_1100_1000_1.0_3000 | 11884 | 11839.40 | 11884 | 11839.40 | 11884 | 11839.40 | 0.00 | 0.475 | |
| bmcp_1100_1000_1.5_3000 | 13468 | 13441.80 | 13468 | 13441.80 | 13468 | 13441.80 | 0.00 | 0.293 | |
| bmcp_1100_1000_2.0_3000 | 120244 | 12000.80 | 120244 | 12000.80 | 120244 | 12000.80 | 0.00 | 0.336 | |

Table 5
Comparison of E2LS with the baseline algorithms on 30 BMCP instances of Set B. Unique/Equal best results appear in bold/italic.

| Instance/Algorithm | Best | Average | Best | Average | Best | Average | Best | Average | S.D. | Time |
|--------------------|------|---------|------|---------|------|---------|------|---------|------|------|
| bmcp_1100_1000_0.3_3000 | 142835 | 242486.70 | 24250 | 142426.00 | 24375 | 24367 | 0.00 | 1.117 | |
| bmcp_1100_1000_0.5_3000 | 11318 | 12509.00 | 11318 | 12509.00 | 11318 | 12509.00 | 0.00 | 0.519 | |
| bmcp_1100_1000_0.75_3000 | 112250 | 112160.20 | 112250 | 112160.20 | 112250 | 112160.20 | 0.00 | 0.452 | |
| bmcp_1100_1000_1.0_3000 | 11884 | 11839.40 | 11884 | 11839.40 | 11884 | 11839.40 | 0.00 | 0.475 | |
| bmcp_1100_1000_1.5_3000 | 13468 | 13441.80 | 13468 | 13441.80 | 13468 | 13441.80 | 0.00 | 0.293 | |
| bmcp_1100_1000_2.0_3000 | 120244 | 12000.80 | 120244 | 12000.80 | 120244 | 12000.80 | 0.00 | 0.336 | |

Zhu et al.: Preprint submitted to Elsevier Page 8 of 12

Efficient and Effective Local Search for the SUKP and BMCP

the SUKP instances with various scales well. For example, ATS-DLA shows good performance for the instances in Sets II and III, since its search operator has a small search region, while a large search region is more important than a high search efficiency for small instances. Algorithms KBTS and MSBTS show good performance for the instances in Set I, because they traverse all the possible moves per step and thus have large search regions. However, KBTS shows worse performance than MSBTS on Set II, since the solution-based tabu search is better than the attributed-based tabu search for the SUKP (Wei and Hao,
Table 6
Comparison of E2LS with the baseline algorithms on BMCP instances of Set C. Unique/Equal best results appear in bold/italic.

| Instance        | Algorithm | Best       | PLTS       | VDLS       |
|-----------------|-----------|------------|------------|------------|
| bmcp_4200_4400_0.3_10000 | E2LS      | 1935947    | 1831638.20 | 1835847.00 |
| bmcp_4200_4400_0.5_7000  | E2LS      | 1778013    | 1747411.90 | 1738553.00 |
| bmcp_4200_4200_0.3_10000 | VDLS      | 1874190    | 1836673.30 | 1738553.00 |
| bmcp_4200_4200_0.5_7000  | E2LS      | 1764277    | 1741211.70 | 1738553.00 |
| bmcp_4200_4400_0.3_10000 | E2LS      | 1954609    | 1929251.00 | 1738553.00 |
| bmcp_4200_4200_0.5_7000  | E2LS      | 1853677    | 1820444.40 | 1738553.00 |
| bmcp_4200_4400_0.3_10000 | E2LS      | 1803465    | 178879.10  | 1738553.00 |
| bmcp_4200_4200_0.5_7000  | E2LS      | 1746064    | 173708.93  | 1738553.00 |

Table 7
Summarized comparisons of E2LS against each baseline algorithm over the three sets of SUKP benchmark instances.

| Algorithm pair | Benchmark | Indicator | #Wins | #Ties | #Losses |
|----------------|-----------|----------|-------|-------|---------|
| E2LS vs. PLTS  | Set I     | Best solution | 3     | 3     | 0       |
| E2LS vs. VDLS  | Set I     | Best solution | 14    | 14    | 2       |
| E2LS vs. KBTS  | Set I     | Best solution | 18    | 12    | 0       |
| E2LS vs. MSBTS | Set I     | Best solution | 18    | 0     | 0       |
| E2LS vs. KBTS  | Set II    | Best solution | 18    | 12    | 0       |
| E2LS vs. MSBTS | Set II    | Best solution | 18    | 0     | 0       |
| E2LS vs. VDLS  | Set II    | Best solution | 29    | 1     | 0       |
| E2LS vs. ATS-DLA| Set II   | Best solution | 29    | 0     | 0       |
| E2LS vs. KBTS  | Set III   | Best solution | 18    | 0     | 0       |
| E2LS vs. MSBTS | Set III   | Best solution | 18    | 0     | 0       |
| E2LS vs. VDLS  | Set III   | Best solution | 16    | 1     | 0       |
| E2LS vs. ATS-DLA| Set III  | Best solution | 16    | 0     | 0       |

2021b). Also, both KBTS and MSBTS show worse performance than ATS-DLA on Set III, since their operators that traverse all the possible moves are inefficient for large instances.

While the proposed E2LS shows excellent performance on all three sets of SUKP instances, because E2LS extracts the advantages of the baseline algorithms and improves their disadvantages. On the one hand, E2LS refines the search region according to the value-weight ratios of the items, so as to abandon the low-quality moves and improve the search efficiency. Thus E2LS works well on Sets II and III. On the other hand, the proposed operator ADD can traverse the refined search region and add the best feasible combination of multiple candidate items into the current solution per iteration, which leads to a large search region. Thus E2LS also works well on Set I. In summary, E2LS is efficient and effective because it can trade-off the search region and search efficiency well.

As for the comparison results on the BMCP instances shown in Tables 4, 5, 6, and 8, the advantage of E2LS over the BMCP baseline algorithms PLTS and VDLS is more obvious than that of E2LS over the SUKP baselines. This might be because the SUKP is more well-studied than the BMCP in terms of heuristic methods. Specifically, E2LS does not lose to the BMCP baselines according to either the best or the average solution indicator, and obtains 7/20 new best-known solutions for the instances in B/C. Moreover, the average solutions of E2LS are equal to its best solutions on
all the tested instances except bmcp_5000_4800_0.5_7000. And E2LS can obtain such excellent results within very small run times for most of the tested instances. The results indicate again the excellent stability, robustness, and efficiency of the proposed E2LS algorithm.

3.4. Analyses on the parameters

We then analyze the influence of the parameters including $r_{num}$, $a_{num}$, and $t$ on the performance of E2LS by comparing E2LS with its variants on each instance set. We first compare the E2LS algorithm with different values of $r_{num}$ and $a_{num}$. We tested five pairs of parameters. They are $r_{num}, a_{num} = 1, 2, 5, 10, m$ respectively. The results are shown in Figure 1, that compares the average values of the best and average solutions of each algorithm on all the instances in each instance set.

The results in Figure 1 show that with the increase of the parameter values from 1 to $m$, the performance of E2LS first increases and then decreases. This result indicates that balancing the search efficiency and the search region is reasonable and necessary. When we set small values to the parameters (e.g., 1 for the SUKP, 1 or 2 for the BMCP), E2LS can explore the search space very quickly, but it might be easy to get stuck in some local optima. When we set large values to the parameters (e.g., 10 or $m$), E2LS can explore the search space widely and deeply, but the low efficiency
Table 9
Comparisons of E2LS, E2LS\(t = 1\), E2LS\(t = 10\), and the baseline algorithms on three sets of SUKP instances. The results are expressed by the average values of the best and average solution, standard deviations, and average run times obtained by each algorithm on all the instances in Sets \(I\) and \(III\).

| Algorithm\Benchmark | Set I |     |     |     |     |     |     |     |     |     |     |     |     |
|---------------------|------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
|                     | Best |     |     |     |     |     |     |     |     |     |     |     |     |
| I2FLS               | 1194.87 | 11938.10 | 25.90 | 90.740 | 9274.47 | 9210.91 | 45.95 | 333.210 | 6583.560 | 6550.470 | 27.91 | 297.920 |
| KBTS                | 11973.60 | 11969.45 | 5.81 | 59.714 | 9348.67 | 9307.43 | 21.31 | 243.472 | 9024.280 | 9005.180 | 23.14 | 147.623 |
| MSBTS               | 11970.03 | 11955.78 | 18.35 | 240.734 | 9361.87 | 9352.69 | 5.78 | 187.351 | 6748.780 | 6819.140 | 130.33 | 104.053 |
| ATS-DLA             | 11932.47 | 11864.17 | 49.69 | 11.693 | 9358.80 | 9357.36 | 0.48 | 39.951 | 9048.890 | 9033.420 | 9.87 | 236.289 |
| E2LS\(t = 1\)      | 11973.60 | 11969.08 | 4.80 | 48.581 | 9362.93 | 9362.55 | 0.15 | 40.110 | 9079.330 | 9075.230 | 5.54 | 235.147 |
| E2LS\(t = 10\)     | 11973.60 | 11969.49 | 6.21 | 43.251 | 9362.93 | 9362.59 | 0.19 | 37.664 | 9079.610 | 9075.190 | 4.70 | 237.549 |
| E2LS                | 11973.60 | 11969.75 | 5.60 | 42.173 | 9362.93 | 9362.60 | 0.19 | 40.235 | 9079.780 | 9075.230 | 4.46 | 227.603 |

Table 10
Comparisons of E2LS, E2LS\(t = 1\), E2LS\(t = 10\), and the baseline algorithms on three sets of BMCP instances. The results are expressed by the average values of the best and average solution obtained by each algorithm, coupled with the average values of the standard deviations and average run times obtained by E2LS, on all the instances in Sets \(B\) and \(C\).

| Algorithm\Benchmark | Set A |     |     |     |     |     |     |     |     |     |     |     |     |
|---------------------|------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
|                     | Best |     |     |     |     |     |     |     |     |     |     |     |     |
| PLTS                | 91533.67 | 91276.79 | – – | – – | 172471.63 | 171196.07 | – – | – – | 196744.97 | 194366.33 | – – | – – |
| VDLS                | 92290.50 | 91980.88 | – – | – – | 174048.87 | 173212.76 | – – | – – | 204202.70 | 203335.67 | – – | – – |
| E2LS\(t = 1\)      | 92290.50 | 92290.50 | 0.00 | 2.508 | 174130.00 | 174130.00 | 0.00 | 20.458 | 204403.65 | 204403.45 | 8.75 | 100.576 |
| E2LS\(t = 10\)     | 92290.50 | 92290.50 | 0.00 | 2.676 | 174130.00 | 174130.00 | 0.00 | 30.668 | 204403.65 | 204403.95 | 1.81 | 70.037 |
| E2LS                | 92290.50 | 92290.50 | 0.00 | 2.676 | 174130.00 | 174130.00 | 0.00 | 30.668 | 204403.65 | 204403.95 | 1.81 | 70.037 |

might also make it hard to escape from local optima.

We can also observe that the properties of the SUKP and BMCP are different. For the SUKP, E2LS with small values of parameters are not good at solving the small instances in Set \(I\), and E2LS with large values of parameters are not good at solving the large instances in Sets \(II\) and \(III\). This result is consistent with the results of the SUKP baseline algorithms, and can explain their performance. That is, algorithms KBTS and MSBTS with large and unrefined search regions are good at solving the small instances in Set \(I\), but do not work well on Set \(III\). In contrast, ATS-DLA with a small and overrefined search region works well on Sets \(II\) and \(III\), but not on Set \(I\).

For the BMCP, the instances in Sets \(A\) and \(B\) prefer small parameter values to large ones. While the instances in Set \(C\) prefer large parameter values to small ones. This might be because for the relatively small instances in Sets \(A\) and \(B\) (with \(S5\) to \(1,600\) items or elements), low search efficiency is more likely to get stuck in local optima than a small search region. While the situation for the large instances in Set \(C\) (with \(4,000\) to \(5,200\) items or elements) is on the contrary. Moreover, the best settings of parameters when solving the SUKP and BMCP are different. The best settings of \(r_{num}\) and \(a_{num}\) are \(2\) and \(5\) for the SUKP and BMCP, respectively. This is also because of different properties of these two problems. For example, when solving the SUKP, items with zero additional weights will not be added into the feasible candidate set \(FC\) (which will be refined by the parameter \(a_{num}\)), and will be added into the current solution directly by the ADD\(^*_\) operator if such a move is not prohibited by the tabu list (lines 5-8 in Algorithm 3). While when solving the BMCP, there is no item with zero additional weights (values), since the value of each item is positive (in the BMCP, an item with zero values should always be contained in the solution). Therefore, if we set the parameter \(a_{num}\) to be the same when solving the SUKP and BMCP, the ADD\(^*_\) operator for the SUKP can add more items than that for the BMCP. Thus it is reasonable to set larger parameter values for the BMCP than for the SUKP.

We further analyze the influence of the parameter \(t\) on the performance of E2LS. We denote E2LS\(t = 1\) and E2LS\(t = 10\) as two variants of E2LS with the parameter \(t\) equals to \(1\) and \(10\), respectively. Note that E2LS\(t = 1\) actually generates the initial solution randomly. We compare these two variants with E2LS and the baseline algorithms. Tables 9 and 10 show the results of the algorithms for the SUKP and BMCP, respectively. The results are the average values of the best and average solutions, the standard deviations, and the average run times of each algorithm on all the instances in each instance set.

As the results show, both E2LS\(t = 1\) and E2LS\(t = 10\) can obtain competitive results with E2LS, and their performance is significantly better than the baseline algorithms for both the SUKP and BMCP. The result indicates that the local search method in E2LS has excellent stability and robustness, as it can yield excellent results with various initial solutions, even if they are generated randomly. Moreover, E2LS slightly outperforms E2LS\(t = 1\) and E2LS\(t = 10\), indicating that higher-quality initial solutions lead to better performance, and the random greedy initialization method in E2LS is effective.
4. Conclusion

This paper proposes an efficient and effective local search algorithm called E2LS for the SUKP and BMCP problems. To our knowledge, this is the first time that an algorithm has been proposed for these two closely related problems. The E2LS algorithm can explore the solution space efficiently by refining the search region, i.e., abandoning the low-quality moves. The proposed ADD* operator in E2LS can traverse the refined search region and provide high-quality moves quickly. As a result, E2LS trades-off the search region and efficiency well, which leads to an excellent performance. Such a trade-off mechanism and the approach of traversing the refined search region could be applied to various combinatorial optimization problems.

Extensive experiments on 78 public SUKP instances and 90 public BMCP instances with various scales demonstrate the superiority of the proposed algorithm. In particular, E2LS provides four new best-known solutions for the SUKP, and 27 new best-known solutions for the BMCP.

Declarations of interest

None.

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.

Cai, S., 2015. Balance between complexity and quality: Local search for minimum vertex cover in massive graphs, in: The Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, pp. 747–753. URL: http://ijcai.org/Abstract/15/111.

Dahmani, I., Ferroum, M., Hifi, M., Sadeghsa, S., 2020. A hybrid swarm optimization-based algorithm for the set-union knapsack problem, in: 7th International Conference on Control, Decision and Information Technologies, CoDIT 2020, pp. 1162–1167. URL: https://doi.org/10.1109/CoDIT49485.2020.926816.

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-tna16704106113>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.

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., Wei, Z., Hao, J., He, K., 2021. Probability learning based tabu search for the budgeted maximum coverage problem. Expert Systems with Applications 183, 115310. URL: https://doi.org/10.1016/j.eswa.2021.115310.

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.2019.06.067.

Özsoydan, F.B., Baykasoglu, A., 2019. A swarm intelligence-based algorithm for the set-union knapsack problem. Future Generation Computer Systems 93, 560–569. URL: https://doi.org/10.1016/j.future.2018.08.002.

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.

Wei, Z., Hao, J., 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.

Zhou, Y., Zhao, M., Fan, M., Wang, Y., Wang, J., 2021. An efficient local search for large-scale set-union knapsack problem. Data Technologies and Applications 55, 233–250. URL: https://doi.org/10.1108/DTA-05-2020-0120.