Adaptive neighborhood reduction-based memetic algorithm for the set-union knapsack problem

Zequn Wei [a] , Jianing Yu [a] , Jin-Kao Hao [b,*] and Jintong Ren [c]

[a] *School of Economics and Management, Beijing University of Posts and Telecommunications, 100876 Beijing, China*

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

[c] *Business School, Hohai University, 210098 Nanjing, China*

Swarm and Evolutionary Computation 101: 102289, 2026

Abstract

The set-union knapsack problem (SUKP) is an important NP-hard variant of the knapsack problem, where each item has a profit and is composed of multiple weighted elements. The SUKP aims to select a subset of items that maximizes the total profit while ensuring the union of their associated elements satisfies the capacity constraint. The SUKP is a challenging combinatorial optimization problem that has been widely studied for its theoretical value and practical relevance. In this work, we present a population-based memetic framework specifically designed for solving the SUKP. The proposed approach integrates an adaptive neighborhood reduction based memetic search, which strengthens intensification by embedding a dynamic profit-to-weight ratio scoring method into a solution-based tabu search. To ensure diversification, a diversity-driven greedy crossover operator and an adaptive population updating rule are developed. The algorithm requires no manual parameter tuning and remains effective across instances of widely varying sizes. Computational results on 132 commonly used benchmark instances demonstrate that our method is both competitive and robust compared with state-of-the-art algorithms. The algorithm is further applied to the SUKP-related budgeted maximum coverage problem, confirming its efficiency and generality. We also provide additional analysis on the influences of several key components of the algorithm.

Keywords: Set-union knapsack problem; Memetic algorithm; Heuristics; Combinatorial optimization.

* Corresponding author.

Email addresses: zequn.wei@bupt.edu.cn (Zequn Wei), yujianing@bupt.edu.cn (Jianing Yu), jin-kao.hao@univ-angers.fr (Jin-Kao Hao), jintong.ren@hhu.edu.cn (Jintong Ren).

1 Introduction

Many decision-making problems in areas such as portfolio selection, network design, data management and manufacturing involve selecting a subset of options from a set of candidates while considering limited resources or capacity constraints. These types of problems can be conveniently modeled as knapsack problems (KP), which are a fundamental and widely studied class of discrete optimization problems with broad practical relevance [1].

Within this family of KP models, the set-union knapsack problem (SUKP) [2] is a widely studied variant that has attracted sustained attention in recent years. A key characteristic of the SUKP is that the weight of any element covered by multiple items is counted only once when evaluating the knapsack capacity constraint. This feature naturally matches many real-world settings and has several practical applications. For example, consider a financial investment problem with multiple investment portfolios where each portfolio is comprised of a set of financial products. Some popular financial products may be listed on various portfolios, so that the common products will not cause extra cost when investing different portfolios. The goal is to determine a set of portfolios to maximize the total expected revenue while keeping the cost of investment within a given budget. This problem can be conveniently modeled by the SUKP where an item corresponds to a financial portfolio with its expected revenue and an element corresponds to a financial product with a projected cost. Thus, solving this financial investment problem is equivalent to finding the optimal solution to the resulting SUKP. The SUKP can also be useful in other relevant applications, including flexible manufacturing [2], cryptography design [3] and data allocation in large cyber systems [4,5]. Meanwhile, the SUKP is a challenging NP-hard problem that combines combinatorial item selection with overlapping element structures. Therefore, advanced metaheuristic solution approaches are necessary to find satisfactory solutions to this problem.

Formally, the SUKP can be defined as follows [2]. Given a set of elements $U = \{1, ..., m\}$ where each element j has a weight $w_j > 0$, a set of items $V = \{1, ..., n\}$ where each item i has a profit $p_i > 0$ and consists of a subset of elements $U_i \in U$. The goal of the SUKP is to find a subset of items S with the maximum total profit while ensuring the total weight of associated elements does not exceed a given knapsack capacity C. A formal formulation of the SUKP is given below:

$$(SUKP) \qquad \text{Maximize} \quad f(S) = \sum_{i \in S} p_i, \tag{1}$$

$$\text{subject to} \quad W(S) = \sum_{j \in \cup_{i \in S} U_i} w_j \leqslant C, \quad S \subseteq V. \tag{2}$$

The SUKP is closely related to several widely studied NP-hard problems. It generalizes the densest k-subhypergraph (DkSH) problem [6] by assigning unit profits to items and unit weights to elements in the SUKP, and by requiring the selection of exactly k items to maximize the number of covered elements. It is also complementary to the budgeted maximum coverage problem (BMCP) [7,8]: the SUKP maximizes item profit under a knapsack constraint on the total weight of the union of covered elements, whereas the BMCP maximizes the profit of covered elements under a budget (knapsack) constraint on the item costs. In addition, the SUKP shares the same covering structure with the budgeted maximum coverage with overlapping costs (BMOC) problem [9], where each item corresponds to a subset of elements and any covered element is counted once. In the BMOC, items carry no intrinsic weight or profit, whereas elements bear both attributes, and the goal is to choose items to maximize the profit of covered elements subject to a budget (knapsack) on their total weight. These connections illustrate both the modeling flexibility and the intrinsic computational difficulty of the SUKP.

In this paper, we focus on advancing and developing metaheuristic algorithms for the NP-hard SUKP. Due to its practical significance, modeling flexibility, and computational complexity, the SUKP has attracted considerable research interest since its introduction in 1994 [2]. As reviewed in Section 2, a number of methods have been proposed for the SUKP, including exact, approximation, and heuristic approaches such as greedy algorithm [10], artificial bee colony algorithm (ABC) [11], local search algorithms [12,13], tabu search algorithms [5,14], particle swarm optimization (PSO) algorithms [15,16] and evolutionary algorithm [17]. However, no existing algorithm consistently performs well across all available SUKP benchmark instances, with the numbers of items and elements ranging from 85 to 5000. Furthermore, population-based methods such as PSO and ABC applied to the SUKP typically rely on continuous to discrete transformations, rather than operating directly in the original binary decision space. To the best of our knowledge, no research has been conducted on the memetic search framework operating directly within the discrete search space. In particular, most existing methods exhibit limitations in both performance and computational efficiency on large-scale instances where the number of items or elements exceeds 1000. This paper addresses these issues by proposing an effective adaptive neighborhood reduction-based memetic algorithm (ANRMA) for solving the SUKP. The main contributions of this work are summarized as follows.

- From the perspective of algorithm design, we propose ANRMA, a memetic algorithm for the SUKP that integrates a solution-based tabu

search enhanced with an adaptive neighborhood reduction strategy to strengthen intensification. We also design a novel diversity-driven greedy crossover operator and an adaptive population updating rule, both dedicated to the SUKP, to enhance diversification. The algorithm handles instances of varying sizes and requires no parameter tuning, making it well-suited for practical applications.

- From the perspective of computational results, we evaluate ANRMA on 132 SUKP benchmark instances with sizes ranging from 85 to 5000 and on 90 BMCP instances, showing that it is robust, competitive, and generalizable. We will make the code of our algorithm publicly available online to facilitate future research.

The remaining part of this paper is structured as follows. In Section 2, we review the existing solution methods for the SUKP in the related literature. Section 3 presents the proposed algorithm framework and its main components in detail. In Section 4, we present computational results on all existing benchmark instances and comparisons with state-of-the-art algorithms. We analyze in Section 5 the convergence of the proposed algorithm and the influences of important components of the algorithm, followed by a general conclusion in Section 6.

2 Related work

The theoretical and practical significance of the SUKP has garnered considerable attention in the literature. A number of algorithms have been proposed for solving the SUKP, including both exact or approximate algorithms and heuristic algorithms.

First, in terms of exact and approximation algorithms, Goldschmidt et al. [2] first studied the SUKP using a general dynamic programming algorithm and showed that it runs in polynomial time when the capacity is polynomial in $m + n$. Later, Arulselvan et al. [10] extended a greedy framework based on an approximation algorithm for the BMCP [7,8] to the case of the SUKP with an additional restriction that each element belongs to at most a constant number d of items. This algorithm achieves a $(1 - e^{-\frac{1}{d}})$ approximation ratio in the studied restricted cases of the SUKP. Generally, these researches only gave theoretical studies of the SUKP without computational experiments. And their proposed algorithms can only run in polynomial time in special cases of the SUKP.

Second, due to the complexity of the SUKP, several metaheuristic algorithms have been proposed to find sub-optimal solutions in a limited time. He et al. [11] designed a binary artificial bee colony algorithm (BABC) for the SUKP

integrating a surjection-based mapping from continuous to binary space with a scoring method S-GROA to greedily repair infeasible solutions. They designed the first set of 30 small benchmark instances with 85 to 500 items and elements and reported the corresponding results. Later, Ozsoydan and Baykasoğlu [15] proposed gPSO, a hybrid binary swarm intelligence algorithm that integrates genetic algorithm operators with particle swarm optimization algorithm to directly explore binary spaces without transfer functions. Lin et al. [18] introduced a hybrid swarm-based optimization algorithm (HBPSO/TS) for the SUKP that combines the binary particle swarm optimization framework with the tabu search algorithm to enhance exploitation capabilities. Wei and Hao [12] proposed an iterated two-phase local search algorithm (I2PLS) based on a local exploration phase and a local optima escaping phase and they also solved the 0/1 linear programming model of the SUKP with the CPLEX solver for the first time and reported the optimal results of six small benchmark instances. Zhou et al. [13] proposed ATS-DLA, an adaptive tabu search for large scale SUKP instances that integrates dynamic item scoring, lightweight neighborhood updates, and age-guided perturbation, and they introduced a new set of 18 large benchmark instances ranging from 850 to 5000 items and elements. Wei and Hao [5] proposed a kernel-based tabu search (KBTS) that combines the tabu search with an intensified kernel search and presented another set of 30 medium benchmark instances with 585 to 1000 items and elements. Later, Wei and Hao [14] proposed a multistart solution-based tabu search algorithm (SBTS) that uses hash vectors as tabu lists to record visited solutions during the search, enabling the algorithm to avoid returning to these previously visited solutions. They also analyzed the impacts of the length and number of the hash vectors on the error rate of the solution-based tabu search procedure. He and Wang [19] introduced a group theory-based optimization algorithm (GTOA) that employs algebraic group operations to drive the evolutionary process and showed the results of applying this algorithm to the SUKP. Dahmani et al. [20] proposed an iterative rounding search (IRSP) that repeatedly solves LP relaxations, fixes the highest fractional item to build a partial solution, and then completes it with a profit over potential weight local greedy solver. They also tested their algorithm on a new set of 54 small to medium benchmark instances with 500 to 1000 items and elements. Then, Dahmani et al. [16] proposed a particle swarm optimization using a backtracking strategy (PSO-B) that employs a path relinking driven backtracking step to intensify exploration between two solutions with high quality as guides. Finally, Zhao and Hifi [17] presented AES-BM, an adaptive evolutionary algorithm based on the scatter search framework featuring in a tailored profit-to-weight scoring based constructive phase with a cooperative variable neighborhood tabu search inside the scatter search framework.

Third, as the rapid development of GPU computing and machine learning, some researchers also combined advanced computing techniques or machine

learning methods with heuristic algorithms to solve the SUKP. Durgut et al. [21] proposed RLABC, a hybrid algorithm integrating a reinforcement learning method (Q-learning) to the artificial bee colony framework to enable adaptive successive operator selection, thereby constructing adaptive operator sequences that enhances search diversity. Ozsoydan and Gölcük [22] proposed a reinforcement learning based hyper heuristic framework to solve the SUKP that employs Q-learning to adaptively select heuristic solvers among PSO, GA, and a genetic based PSO according to their historical performance. Zhao et al. [23] designed a hybrid algorithm combining an incremental Q-learning module with a variable neighborhood tabu search framework. Then, Sonu and Ozcan [24] proposed a CUDA-based parallel local search method for the SUKP applying the GPU parallel computing architecture to a local search driven by popular crossovers. Other researchers [25,26,27] also apply several heuristics and machine learning methods to solve the SUKP.

Although a variety of heuristics and learning-based solution approaches have been proposed for the SUKP in recent years, their performance is often limited to small or medium sizes of instances. Moreover, there remains a need to further improve the computational efficiency of SUKP algorithms to meet practical requirements, particularly for large-scale instances. In this study, we introduce an adaptive neighborhood reduction-based memetic algorithm that achieves competitive results across all known instance sizes while maintaining high computational efficiency.

3 Adaptive neighborhood reduction-based memetic algorithm for the SUKP

In this section, we present the adaptive neighborhood reduction-based memetic algorithm (ANRMA) for the SUKP. The memetic algorithm [28,29] is a hybrid heuristic optimization approach that combines evolutionary computation and local optimization, and has been successfully applied to solve various combinatorial optimization problems. Our ANRMA algorithm consists of three main components: a diversity-driven greedy crossover operator, an efficient solution-based tabu search with adaptive neighborhood reduction, and an adaptive population updating rule that considers both solution quality and diversity. We first introduce its main framework and then explain its components.

3.1 Main framework

The main procedure of the ANRMA algorithm is described in Algorithm 1. The ANRMA first constructs the initial population P by generating a set of feasible solutions (line 3) and records the best solution found so far in S^* (line 4). Then, the algorithm enters a 'while' loop (lines 5-12) to search for improved solutions while evolving the population. At each iteration, a dedicated diversity-driven greedy crossover operator is applied to generate a new offspring solution S_o based on the current population (line 6). Then the offspring is refined using the solution-based tabu search with adaptive neighborhood reduction (line 7), aiming to explore diverse high-quality local optima. After updating the best solution S^* found so far (lines 8-10), the population is adaptively updated according to the solution quality and diversity of the current local optimum S_b (line 11). Finally, the whole algorithm terminates and returns the overall best solution found S^* when the given time limit t_{max} is reached.

Algorithm 1 Adaptive neighborhood reduction-based memetic algorithm for the SUKP

1: **Input**: Instance I, cutoff time t_{max}, maximum number of consecutive non-improving iterations λ_{max}.
2: **Output**: The overall best solution found S^*.
3: $P \leftarrow Population\_Initialization(I)$ /* See Algorithm 2 */
4: $S^* \leftarrow Record\_Best\_Solution(P)$
5: **while** $Time \leq t_{max}$ **do**
6: $S_o \leftarrow Diversity\_Driven\_Greedy\_Crossover(P)$ /* See Algorithm 5 */
7: /* See Algorithm 3 */
$S_b \leftarrow Tabu\_Search\_With\_Neighborhood\_Reduction(S_o, \lambda_{max})$
8: **if** $f(S_b) > f(S^*)$ **then**
9: $S^* \leftarrow S_b$
10: **end if**
11: $P \leftarrow Adaptive\_Population\_Updating(S_b, P)$ /* See Algorithm 6 */
12: **end while**
13: **return** S^*

3.2 Population initialization

As shown in Algorithm 2, the population initialization procedure begins by setting the population size according to $PopSize = n/1000 + 5$ (line 3), where n is the number of items of the SUKP instance. The constant term ensures a reasonable minimum population size for small instances, preventing insufficient diversity when n is small. As a result, the population size increases with the instance size, helping to maintain population diversity and improve the overall search effectiveness. Subsequently, the population P

is initialized (line 4), followed by an iterative procedure that populates it with initial solutions (lines 5–9). In each iteration, a feasible initial solution S is generated at random (line 6) by selecting items subjected to the capacity constraint. Then the solution S is refined by a short-run of the solution-based tabu search procedure with adaptive neighborhood reduction (SBTS, line 7) as described in Section 3.3. Then, the improved solution S is added to the population P (line 8). The entire initialization procedure is repeated until the number of solutions in the population reaches $PopSize$.

Algorithm 2 Population initialization

1: **Input**: Instance I.
2: **Output**: The initial population P.
3: $PopSize \leftarrow Adaptive\_Population\_Size(I)$ /* Determine $PopSize$ adaptively */
4: $P \leftarrow \emptyset$
5: **while** $|P| \leq PopSize$ **do**
6: $S \leftarrow Random\_Initialize(I)$
7: /* Run short SBTS to improve S. See Algorithm 3* / $S \leftarrow Tabu\_Search\_With\_Neighborhood\_Reduction(S, \lambda_{max1})$
8: $P \leftarrow Add\_One\_Solution(S, P)$ /* Add solution S into population */
9: **end while**
10: **return** P

3.3 Solution-based tabu search with adaptive neighborhood reduction

Tabu search (TS) [30] is widely regarded as a well-suited method for solving 0-1 decision-making problems. The core strategy of TS is to manage a memory data structure called the tabu list to record the information of the local search process to avoid returning to previously visited solutions. Many existing algorithms for the SUKP incorporate a TS component, often with problem-specific adaptations to improve performance. Moreover, with a dedicated tabu list design, solution-based tabu search (SBTS) [14] can record visited solutions and directly prevent revisiting them, and has been shown to be a powerful method for the SUKP. We adopt the SBTS procedure to improve the offspring solutions generated by the crossover operator. However, due to the high computational cost of evaluating neighborhood solutions, the performance of SBTS becomes limited as the instance size increases. To address this issue, we propose an adaptive neighborhood reduction strategy to significantly enhance search efficiency.

The main scheme of our SBTS with adaptive neighborhood reduction is shown in Algorithm 3. The SBTS procedure first initializes several components based on the current solution S, including the indicator $Find$ for detecting admissible neighboring solutions (line 3), the counter λ for

Algorithm 3 Solution-based tabu search with adaptive neighborhood reduction

1: **Input**: Offspring solution S_o or initial solution S, maximum number of consecutive non-improving iterations λ_{max}.
2: **Output**: The best solution S_b found during the SBTS procedure.
3: $Find \leftarrow True$
4: $\lambda \leftarrow 0$
5: $H_{1\text{-}3} \leftarrow Initialize\_Hash\_Vectors$
6: $S_b \leftarrow S$
7: **while** $\lambda < \lambda_{max} \wedge Find$ **do**
8: /* Construct the reduced neighborhood adaptively. See Section 3.3.1 */
$N_r(S) \leftarrow Adaptive\_Neighborhood\_Reduction(S)$ /* See Algorithm 4 */
9: Find admissible neighboring solutions $N'(S)$ in $N_r(S)$
10: **if** $N'(S) \neq \emptyset$ **then**
11: $S \leftarrow argmax\{f(S') : S' \in N'(S)\}$
12: $Find \leftarrow True$
13: **else**
14: $Find \leftarrow False$
15: **end if**
16: **if** $f(S) > f(S_b)$ **then**
17: $S_b \leftarrow S$
18: $\lambda \leftarrow 0$
19: **else**
20: $\lambda \leftarrow \lambda + 1$
21: **end if**
22: $H_{1-3} \leftarrow Update\_Hash\_Vectors(H_{1-3}, S)$
23: **end while**
24: **return** S_b

tracking consecutive non-improving iterations (line 4), the hash vectors for tabu list management (line 5), and the best solution S_b found during SBTS (line 6). Then, the SBTS procedure iteratively searches the reduced neighborhood to improve the current solution S (lines 7–23). Specifically, in each iteration of the 'while' loop, the reduced neighborhood $N_r(S)$ is first constructed based on the current solution S (line 8, see also Algorithm 4). Next, the admissible neighboring solutions $N'(S)$ in $N_r(S)$ are identified (line 9), and the best one S' is selected to update S (lines 10–15). Whenever an improved solution is found, the best solution S_b is updated accordingly (lines 16-21), and the hash vectors are updated at the end of each iteration (line 22). The SBTS procedure terminates when either the maximum number of consecutive non-improving iterations λ_{max} is reached or no admissible neighboring solution S' can be found in the reduced neighborhood $N_r(S)$ (i.e., $N'(S) = \emptyset$), where λ_{max} is an adaptive parameter determined by the items size n as $\lambda_{max} = n/8$. Note that in each SBTS iteration, the best non-tabu neighboring solution S' may be worse than the current solution S; however, it is still accepted to replace S in order to

continue the search. This strategy helps the algorithm escape local optima and explore new regions of the search space in pursuit of better solutions.

3.3.1 Adaptive neighborhood reduction

We adopt the *swap* move operator, which replaces items in the current solution to explore improved neighboring solutions. However, as the instance size increases, the number of potential combinations generated by the *swap* operator grows significantly (quadratic complexity), greatly expanding the search space and increasing computational overhead. To address this issue, an adaptive neighborhood reduction (ANR) strategy (see line 8, Algorithm 3) is employed to refine the set of candidate items, thereby reducing the search space and enhancing exploitation. This approach accelerates the search process and improves efficiency, particularly for large-scale instances.

Given a solution $S = \langle A, \bar{A} \rangle$, items in A are considered for removal and items in $\bar{A}$ for addition by the *swap* operator. Generally, the ANR strategy aims to reduce the size of A and $\bar{A}$ by limiting the maximum number of candidate items to remove or add. The key to this strategy lies in identifying promising candidates for constructing the reduced neighborhood. In our case, an adaptive procedure is employed based on the dynamic profit-to-weight ratio, as introduced in Section 3.3.2.

Algorithm 4 Adaptive neighborhood reduction

1: **Input**: Input solution S.
2: **Output**: Reduced neighborhood $N_r(S)$.
3: $N_r(S) \leftarrow \emptyset$
4: /* n_r is the maximum number of candidate items to remove, and n_a is the maximum number of candidate items to add */
$(n_r, n_a) \leftarrow Adaptive\_Setting\_Parameters(S)$
5: /* Find the set A' of the top n_r selected items with the lowest dynamic profit-to-weight ratio R_D^- (see Section 3.3.2) */
$A' \leftarrow \text{argmin}_{x \in A}^{n_r} \ R_D^-(x \mid S)$
6: **for** each item $x \in A'$ **do**
7: $N_x \leftarrow \emptyset$ /* N_x is the reduced neighborhood induced by removing item x */
8: /* Find the set $\bar{A}'_x$ of the top n_a items with the highest dynamic profit-to-weight ratio R_D^+ (see Section 3.3.2) */
$\bar{A}'_x \leftarrow \text{argmax}_{y \in \bar{A}}^{n_a} \ R_D^+(y \mid S \setminus \{x\})$
9: $N_x \leftarrow Get\_Reduced\_Neighborhood(A' \setminus \{x\}, \bar{A}'_x)$
10: $N_r(S) = N_r(S) \cup N_x$
11: **end for**
12: **return** $N_r(S)$

As shown in Algorithm 4, the ANR procedure first initializes an empty neighborhood $N_r(S)$ (line 3) and adaptively sets two parameters to control the reduction size (line 4), including the maximum number of candidate

items to remove n_r (set to $min\{round(e^{-6\times10^{-4}n}) + 1, |A|\}$) and the maximum number of candidate items to add n_a (set to $min\{n_r + 1, |\bar{A}|\}$). The reduced selected item set A' is then constructed (line 5) by selecting the top n_r items in A with the lowest dynamic profit-to-weight ratio R_D^- values, as defined in Equation (6). Considering that removing different items in A' may affect the dynamic profit-to-weight ratio R_D^+ (see Equation (7)) of non-selected items due to changes in the covered elements, we construct the reduced non-selected item set $\bar{A'}_x$ with respect to each item $x \in A'$. Through this dynamic item scoring mechanism, a more promising reduced neighborhood $N_r(S)$ is identified iteratively (lines 6–11). Specifically, after initializing the reduced neighborhood N_x with respect to item x (line 7), we compute R_D^+ according to $S \setminus \{x\}$, which represents a hypothetical solution obtained by removing item x from S. Then, we select the top n_a items in $\bar{A}$ with the highest R_D^+ values as the items of the reduced non-selected set $\bar{A'}_x$ (line 8). Then, the reduced neighborhood N_x associated with item x is constructed based on the reduced item sets $A' \setminus \{x\}$ and $\bar{A'}_x$ (line 9), and is subsequently recorded as part of the overall reduced neighborhood $N_r(S)$ (line 10). Finally, when all items in A' have been examined, the ANR procedure terminates and returns the overall reduced neighborhood $N_r(S)$.

Given the reduced selected item set A' and non-selected set $\bar{A'}_x$, the *swap* operator is employed to construct the reduced neighborhood. The $swap(x, y)$ operator removes an item x in A' then adds an item y in $\bar{A}'_x$ while respecting the knapsack capacity constraint and not resulting in a tabu solution. Meanwhile, the $swap(x, y)$ operator also allows only removing an item $x \in A'$ or only adding an item $y \in \bar{A'}_x$. To represent such single-item operations, we introduce a dummy item i^* into the swap operator. Generally, the reduced neighborhood $N_r(S)$ induced by *swap* can be defined as follows:

$$N_r(S) = \{S' \mid S \oplus swap(x, y) : x \in A' \cup \{\, i^*\}, y \in \bar{A}'_x \cup \{\, i^*\}, \sum_{j \in \cup_{i \in S'} U_i} w_j \leqslant C\}. \tag{3}$$

Before the reduction, the neighborhood created by the *swap* operator comprises all combinations of exchanging, adding and removing items in A and $\bar{A}$, leading to a time cost of $O_1 = n + |A| \times (n - |A|)$ for traversing the whole neighborhood with the time complexity of $O(n^2)$. However, for our ANR procedure, the time cost of traversing the neighborhood can be reduced to $O(n_r n_a)$, which tends to a constant time complexity $O(1)$ as the growth of item size n. Thus, this ANR strategy can significantly reduce the time cost of examining the neighboring solutions especially on large-scale instances. We analyze the effectiveness of our ANR procedure in Section 5.2.

3.3.2 Dynamic profit-to-weight ratio

We adopt the dynamic profit-to-weight ratio (R_D) [13,14,17] to identify promising candidate items when constructing the reduced neighborhood. Unlike the static profit-to-weight ratio commonly used in solving the SUKP, which assigns each item a fixed value based solely on the given instance, R_D is computed adaptively with respect to the current solution S, taking into account the elements already covered by the selected items. Consequently, the ratio of an item may vary across different solutions. This dynamic evaluation leverages the fact that each element's weight is counted only once, thereby providing a more accurate assessment of an item's contribution to the current solution.

We first define the marginal weight ΔW to capture the dynamic effect of adding or removing an item on the total weight of the current solution. Given a solution $S = (A, \bar{A})$, where item x belongs to the selected set A and item y belongs to the non-selected set $\bar{A}$, the marginal weight ΔW^- for removing x from S and ΔW^+ for adding y to S can be defined as follows:

$$\Delta W^-(x \mid S) = \sum_{j \in U_x \wedge j \notin \cup_{i \in S \setminus \{x\}} U_i} w_j, \tag{4}$$

$$\Delta W^+(y \mid S) = \sum_{j \in U_y \wedge j \notin \cup_{i \in S} U_i} w_j, \tag{5}$$

where U_x and U_y denote the sets of elements covered by items x and y, respectively. In Equation (4), $\cup_{i \in S \setminus \{x\}} U_i$ represents the set of elements covered by $S \setminus \{x\}$, i.e., the remaining elements after removing item x. In Equation (5), $\cup_{i \in S} U_i$ denotes the set of elements covered by the solution S.

The dynamic profit-to-weight ratio R_D^- for removing x from S and R_D^+ for adding y to S are then defined as follows:

$$R_D^-(x \mid S) = p_x / \Delta W^-(x \mid S), \tag{6}$$
$$R_D^+(y \mid S) = p_y / \Delta W^+(y \mid S), \tag{7}$$

where p_x and p_y represent the profits of items x and y, respectively.

3.3.3 Tabu list management

In the SBTS procedure, the tabu list is implemented using three binary hash vectors H_k ($k = 1, 2, 3$) to record previously visited neighboring solutions [14]. To compute the hash value of each neighboring solution, we use the

hash functions $h_k(S) = \left(\sum_{i=1}^{m} \lfloor \mathcal{W}_i^k \times y_i \rfloor\right) \bmod L$, where $\mathcal{W}_i^k$ are pre-computed weight vectors and L is the fixed length of the hash vectors. When a new solution S is found during the search, the three hash values aforementioned are first calculated as the indexes of the respective hash vectors. If all hash vector values are 1, i.e., $H_1[h_1(S)] \wedge H_2[h_2(S)] \wedge H_3[h_3(S)] = 1$, the solution S is classified as visited and is no more considered during the subsequent search; otherwise, S is considered a non-tabu neighboring solution. In this way, the SBTS procedure can determine the tabu status of each solution found during the search in $O(1)$ time, enabling rapid local search with high efficiency and effectiveness.

3.4 Diversity-driven greedy crossover

Crossover operator is a key component in a memetic algorithm, which generates new offspring solutions by inheriting good features from its parent solutions. For knapsack problems, a common approach is to construct the offspring from the items selected in the parent solutions, such as the backbone-based crossover [31,32]. However, incorporating certain items that are not present in either parent solution can also enhance population diversity and potentially lead to better exploration of the solution space. Unlike the backbone-based crossover, we design a tailored diversity-driven greedy crossover (DGC) operator that allows items not selected by the parent solutions to be accepted in an offspring solution. Each item is assigned a selection probability derived from the parents. Then all items are divided into a promising set and a less-promising set. Finally, promising items are used to construct the offspring solution according to their dynamic profit-to-weight ratios (see Section 3.3.2).

As shown in Algorithm 5, after initializing the offspring solution S_o (line 3) and randomly selecting the parent solutions S_a and S_b (line 4), the DGC crossover proceeds in three main steps. In Step 1, we calculate the selection probability of each item using Equation (8) (line 5), which considers how many times the item is chosen in the parent solutions. The items are then divided into two candidate sets (line 6): V_p, containing promising items with higher probabilities, and $V_l = V \setminus V_p$, containing less-promising items, where V denotes the entire item set. In Step 2 (lines 7-14), we sequentially add items from V_p to the offspring solution S_o based on their dynamic profit-to-weight ratios (see Section 3.3.2) with respect to S_o until no further items can be added. In Step 3 (lines 15-22), we iteratively identify feasible items from V_l and add the one with the highest dynamic profit-to-weight ratio to S_o, until no feasible candidate remains. Finally, the resulting offspring solution S_o is returned as the output of the DGC crossover.

Algorithm 5 Diversity-driven greedy crossover

Input: Population P.
Output: Offspring solution S_o.
$S_o \leftarrow \emptyset$
Randomly choose two different solutions S_a and S_b in P
/* **Step 1**: Calculate item selection probabilities and get candidate item sets */
$\alpha \leftarrow Calculate\_Probability(S_a, S_b, P)$
$(V_p, V_l) \leftarrow Select\_Candidate\_Item(\alpha)$
while $W(S_o) < C$ **do**
  /* **Step 2**: Add item from promising set V_p to S_o */
  $i \leftarrow \text{argmax}_{i \in V_p} \ R_D^+(i \mid S_o)$ /* See Section 3.3.2 */
  if $W(S_o \cup \{i\}) \leq C$ **then**
    $S_o \leftarrow S_o \cup \{i\}$, $V_p \leftarrow V_p \backslash \{i\}$
  else
    break
  end if
end while
while V_l is not empty **do**
  /* **Step 3**: Add item from less-promising set V_l to S_o */
  $V_l' \leftarrow Find\_Feasible\_Item(V_l, S_o)$/*Find feasible items in V_l to add to S_o*/
  if V_l' is not empty **then**
    $j \leftarrow \text{argmax}_{j \in V_l'} \ R^+(j \mid S_o)$ /* See Section 3.3.2 */
    $S_o \leftarrow S_o \cup \{j\}$, $V_l \leftarrow V_l \backslash \{j\}$
  else
    break
  end if
end while
return S_o

Given a solution S where $S[i] = 1$ indicates that item i is selected and $S[i] = 0$ otherwise, all items with respect to the two parent solutions S_a and S_b are classified into three categories in Equation (8): (i) $S_a[i]+S_b[i] = 2$, representing the items selected by both parents; (ii) $S_a[i]+S_b[i] = 1$, representing the items selected by only one parent; and (iii) $S_a[i] + S_b[i] = 0$, representing the items not selected by either parent. The DGC crossover then generates the offspring according to each item's selection probability α_i, defined as follows:

$$\alpha_i = \begin{cases} 0.1 + 0.4e^{-0.04\overline{D}}, & S_a[i] + S_b[i] = 0, \\ 0.5, & S_a[i] + S_b[i] = 1, \\ 1 - (0.1 + 0.4e^{-0.04\overline{D}}), & S_a[i] + S_b[i] = 2, \end{cases} \tag{8}$$

where $\overline{D} = \frac{\sum D}{|P|}$ denotes the average Hamming distance of all solution pairs in the population P, and D is the Hamming distance between any two different

solutions in P. In this setting, items selected by both or by neither parent are assigned complementary probabilities of inclusion in candidate item set V_p, adaptively determined by the population diversity $\overline{D}$. As diversity decreases, the probability for unselected items increases to promote exploration, whereas a higher diversity favors the selection of common items.

Note that in Steps 2 and 3, items are added to the offspring solution S_o in a greedy way. Unlike the static profit-to-weight ratio (denoted by S-GROA) used in [11], we adopt the dynamic profit-to-weight ratio introduced in Section 3.3.2, which allows items to be selected from V_p or V_l according to the current state of S_o, thereby facilitating the construction of higher-quality offspring solutions.

3.5 Adaptive population updating

We develop an adaptive population update mechanism that adaptively balances quality and diversity. The main steps of the adaptive population updating procedure are outlined in Algorithm 6. First, the best solution S_b obtained during the SBTS procedure is inserted into the current population P (line 3). Next, each solution in P is evaluated using the adaptive goodness score ϕ_S defined in Equation (9) (line 4). Finally, the solution with the lowest goodness score is removed (lines 5–6), and the updated population P is returned (line 7).

Algorithm 6 Adaptive population updating

1: **Input**: Best solution S_b found during SBTS, current population P.
2: **Output**: Updated population P.
3: $P \leftarrow P \cup S_b$
4: $\phi_s(P) \leftarrow Score(P)$ /* Score all solutions in P based on Equation (9) */
5: Sort solutions in P in descending order by $\phi_S(P)$
6: $P \leftarrow Remove\_Last\_Solution(P)$
7: **return** P

The proposed adaptive goodness scoring strategy evaluates each candidate solution by jointly taking into account both quality and diversity. Specifically, the quality of a solution is measured by its objective value, while its diversity with respect to the population is determined by the minimal Hamming distance to the other members. Both quality and diversity are normalized using the Max–Min method and then aggregated with different weights, as defined below:

$$\phi_S = \beta \times \frac{f(S) - f_{min}}{f_{max} - f_{min}} + (1 - \beta) \times \frac{D(S) - D_{min}}{D_{max} - D_{min}}, \tag{9}$$

where f_{min} and f_{max} denote the minimal and maximal objective values in the population, D_{min} and D_{max} represent the minimal and maximal distances measured by the minimal Hamming distance of solution S to the other solutions, $f(S)$ and $D(S)$ are the objective value and distance of S, respectively, and β is a weight parameter in $[0, 1]$.

The weight parameter β plays a crucial role in balancing quality and diversity in the adaptive goodness scoring strategy. Unlike the scoring strategy, where β is set empirically (e.g., in [33]), we design an adaptive mechanism that adjusts β dynamically according to the current status of the population. Specifically, β is defined as follows:

$$\beta = \frac{f(S_o) - f_{min}}{f_{max} - f_{min}}, \tag{10}$$

where $f(S_o)$ is the objective value of the offspring solution S_o. As a result, β increases when $f(S_o)$ is relatively high, encouraging the population to accept high-quality solutions, and decreases when $f(S_o)$ is less competitive, thereby emphasizing diversity and maintaining an adaptive balance between solution quality and population diversity.

3.6 Complexity analysis

Given a SUKP instance with n items and m elements, we analyze the computational complexity of ANRMA by examining its four major components.

First, the population initialization procedure (see Section 3.2) constructs the initial population of size $P = \Theta(n)$, as the population size grows linearly with the number of items. For each individual, a random initialization is first performed in $O(nm)$ time, followed by a short-run SBTS procedure in $O(nm)$ time. Therefore, the initialization of the entire population requires $O(P \cdot nm) = O(n^2 m)$ time.

Second, the SBTS with adaptive neighborhood reduction procedure (see Section 3.3) improves the input solution until the maximum number of consecutive non-improving iterations λ_{max} is reached. During each iteration, the ANR strategy is able to explore the reduced neighborhood in a time tending to $O(1)$ (see Section 3.3.1) and execute the best move in $O(nm)$ time. Thus, the computational complexity of each iteration is $O(nm)$, resulting in an overall complexity of $O((n + \lambda_{max}) \cdot nm) = O(n^2 m)$.

Third, in the diversity-driven greedy crossover procedure (see Section 3.4),

two parents generate a new offspring solution by dividing promising item sets and inserting valuable items in $O(n^2m)$ time due to the evaluations of the dynamic profit-to-weight ratio across the items.

Finally, the adaptive population updating procedure (see Section 3.5) merges the current population sized $P = \Theta(n)$ with the new offspring and evaluates pairwise hamming distances. Since each distance computation requires $O(n)$, the overall complexity of this step is $O((P+1)^2 n) = O(n^3)$.

After initialization, each outer iteration of ANRMA performs crossover, SBTS procedure, and population updating. The overall complexity is therefore $O(n^2m) + G\Big(O(n^2m) + O(n^2m) + O(n^3)\Big)$, where G is the number of outer iterations (bounded by the cutoff time t_{max}). Since the SUKP benchmarks satisfy $m = \Theta(n)$, the overall worst-case complexity simplifies to $O(n^3) + G \cdot O(n^3) = O(Gn^3)$.

4 Computational results and comparisons

This section presents a performance evaluation of the proposed ANRMA algorithm. We present computational results on three sets of 132 benchmark instances commonly used in the literature and compare the performance with state-of-the-art algorithms for each set.

4.1 Benchmark instances

The SUKP benchmark instances used in our experiments span a wide range of item sizes and are categorized into three sets as follows. (i) Set I consists of 60 benchmark instances of small to medium size, with 85 to 1000 items and elements. This set was proposed in [11] and [5] and has been widely used in most studies on the SUKP, as noted in Section 2 [1]. (ii) Set II comprises 54 instances with 500 to 1000 items and elements, which were designed by [20] and have been commonly evaluated in [25,20,26,16,23,17]. (iii) Set III is composed of 18 large instances introduced in [13], ranging from 850 to 5000 items and elements [2].

Each instance, denoted as $n\_m\_\alpha\_\beta$, is defined by four parameters. n and m denote the numbers of items and elements, respectively. A relation matrix

[1] The 60 benchmark instances of Set I are available at: `http://www.info.univ-angers.fr/pub/hao/SUKP_MSBTS.html`

[2] The 72 benchmark instances of Set II and III are available at our GitHub link: `https://github.com/Zequn-Wei/SUKP`

$R_{ij} \in 0, 1^{m \times n}$ indicates the association of items and elements, where $R_{ij} = 1$ indicates that item i covers element j. The parameter α is the density of R, i.e., $\alpha = \left(\sum_{i=1}^{n} \sum_{j=1}^{m} R_{ij}\right)/mn$, and β is the ratio between the knapsack capacity C and the total element weight, i.e., $\beta = C/\sum_{j=1}^{m} w_j$.

Given the large number of instances, we aggregated the results of Set I and Set II into 19 groups by averaging values according to item and element sizes. Specifically, every six instances in Set I with the same item size or element size were assigned to one group, yielding 10 groups from the 60 instances in Set I. For example, group I-100 includes instances 100_85_0.10_0.75, 100_85_0.15_0.85, 100_100_0.10_0.75, 100_100_0.15_0.85, 85_100_0.10_0.75 and 85_100_0.15_0.85. Similarly, the 54 instances in Set II are grouped into nine sets, where each group consists of six instances with identical item and element sizes. For example, group II-500_600 includes instances 500_600_0.10_0.70, 500_600_0.15_0.70, 500_600_0.10_0.75, 500_600_0.15_0.75, 500_600_0.10_0.80 and 500_600_0.15_0.80.

4.2 Experimental settings

Reference algorithms. For each set of instances, we use three state-of-the-art algorithms from the literature for comparison purposes. For Sets I and II, we compare our algorithm with the multistart solution-based tabu search algorithm [14](MSBTS), the particle swarm optimization using backtracking [16](PSO-B), and the adaptive evolutionary search-based method [17](AES-BM). For Set III, we use the following reference algorithms: MSBTS [14], the iterated two-phase local search (I2PLS) [12], and the adaptive tabu search with dynamic list adjustment (ATS-DLA) [13].

We run I2PLS [12] and MSBTS [14] under the same experimental settings as our ANRMA algorithm to obtain the computational results of these reference algorithms. For PSO-B [16] and AES-BM [17], for which the code is unavailable, we cite the results reported in [17], which only include the results of the first 30 instances in Set I. The results of ATS-DLA are also from the corresponding paper [13] with 10 independent runs with a cutoff time of 1000 seconds for each instance. Additionally, we also present the results obtained by the CPLEX solver (version 22.1.1) for all three sets of instances under a time limit of two hours using the general 0/1 linear programming approach.

Computing platform. The proposed ANRMA algorithm is coded in C++ and compiled by the g++ compiler with the `-O3` option. All the experiments were carried out on an Intel Xeon Gold 6148 processor with 2.4 GHz CPU

and 2 GB RAM running under the Linux operating system. We provide the summary of the computing platforms used for the references in Table 1. To ensure a fair comparison, our experiments were conducted on hardware with a CPU comparable to the lowest specifications reported in the reference studies. Moreover, both our implementation and the available implementations of the reference algorithms are written in C++ and executed under Linux environments, which helps reduce potential bias arising from differences in programming languages or operating systems. The algorithm will be publicly available at the GitHub link of footnote 2.

Table 1
Summary of the computing environments used for ANRMA and the references.

Algorithm	Language	Processor	CPU (GHz)	RAM (GB)	System
MSBTS [14]	C++	Intel Xeon Gold 6148	2.4	2	Linux
PSO-B [16]	C++	Intel Pentium Core i5	2.4	-	-
AES-BM [17]	C++	Intel Pentium Core i7	3.6	32	Linux
I2PLS [12]	C++	Intel Xeon Gold 6148	2.4	2	Linux
ATS-DLA [13]	-	Intel Xeon E5-2640	2.4	128	Linux
ANRMA (this work)	C++	Intel Xeon Gold 6148	2.4	2	Linux

Stopping conditions. To ensure a fair comparison, the stopping condition for our algorithm is set in accordance with the settings used in the literature. Specifically, each instance of Sets I and III is independently solved 100 times with the cutoff time of 500 seconds for instances comprising up to 500 items, and 1000 seconds for instances having over 500 items. For Set II, each instance is solved 10 times with the cutoff time of 500 seconds for instances with up to 500 items, and 1000 seconds for instances with over 500 items. All repeated runs are executed using different but fixed random seeds to ensure controlled randomness.

Parameter settings. The ANRMA algorithm incorporates four parameters that are adaptively determined by the instance size or the state of the current solution during the search. Table 2 summarizes the parameter settings, where n denotes the number of items in a SUKP instance. Note that the constant terms in $PopSize$ and λ_{max} ensure stable behavior across different instance sizes. The offset in the population size avoids unrealistically small populations for very small instances, while both expressions still allow the parameters to grow adaptively with n. Our preliminary experiments indicate that the algorithm is not sensitive to these constants, and alternative values lead to similar performance, confirming the robustness of this parameterization. These parameter settings are used as defaults throughout the experiments.

Table 2
Summarized parameter settings of the ANRMA algorithm.

Parameter	Section	Setting	Description
$PopSize$	3.2	$n/1000 + 5$	Population size
λ_{max}	3.3	$n/8$	Maximum consecutive iterations
n_r	3.3.1	$min\{round(e^{n\times-6e-4}) + 1, \lvert A\rvert\}$	Reduced size of candidate items
n_a	3.3.1	$min\{n_r + 1, \lvert\bar{A}\rvert\}$	Reduced size of candidate items

4.3 Computational results on the SUKP benchmark instances

The aggregated comparison results for Sets I and II are reported in Table 3, while the detailed results for each instance in Sets I and II are given in Table A.1 and Table A.2 of Appendix A, respectively. The results for each instance in Set III are presented in Table 4. The summarized comparison between ANRMA and the reference algorithms on the three benchmark sets is reported in Table B.1 of Appendix B. Solution certificates for all instances are available at the GitHub link of footnote 2.

In Tables 3 and 4, as well as Tables A.1 and A.2 in Appendix A, we show the following information. The first column gives the name of each instance or aggregated instance group. The second column presents the best-known values (BKV) reported in previous studies. The third and fourth columns report the best lower bound (LB) and upper bound (UB) obtained by the CPLEX solver (version 22.1.1). The remaining columns present the following indicators: the best objective value (f_{best}), the average objective value (f_{avg}), the standard deviation (std) over all runs (100 runs for Sets I and III, 10 runs for Set II), and the average runtime (t_{avg} in seconds) required to obtain the f_{best} value of our algorithm. A dash (–) denotes that the corresponding results are unavailable and the row $\#Avg$ reports the average value of each column. Moreover, the best f_{best} and f_{avg} results in each row are highlighted in bold, while equal best results are indicated in *italic*.

From Table 3 and Table A.1, we can observe that our ANRMA algorithm attains all the six optimal solutions confirmed by the CPLEX solver (marked with an asterisk (*) in Table A.1) and reaches all the BKV results on Set I (small to medium instances). Moreover, ANRMA provides better f_{avg} results for seven out of the ten groups, indicating its high stability compared to the reference algorithms. Importantly, on Set II (medium instances), ANRMA finds 12 new best-known f_{best} results (see Table A.2) and attains better f_{best} values for four out of the nine groups in Table 3. ANRMA also dominates the reference algorithms on Set II in terms of the f_{avg} results. From Table 4, we observe that on Set III (large instances), CPLEX fails to find a feasible solution for most instances, whereas ANRMA improves two BKV results and matches

Table 3
Computational results and comparisons of the ANRMA algorithm with reference algorithms on the SUKP benchmark instance of Set I and Set II (small to medium size, 114 instances aggregated into 19 groups, with each row representing the average over six instances).

Instance group	BKV	CPLEX		MSBTS [14]		PSO-B [16]		AES-BM [17]		ANRMA (this work)			
		LB	UB	f_{best}	f_{avg}	f_{best}	f_{avg}	f_{best}	f_{avg}	f_{best}	f_{avg}	std	t_{avg}
I-100	12954.67	12954.67	12954.67	*12954.67*	12945.92	*12954.67*	*12954.67*	*12954.67*	*12954.67*	*12954.67*	*12954.67*	0	0.80
I-200	12928.17	10747.83	28670.42	*12928.17*	12888.59	*12928.17*	12920.12	*12928.17*	12921.83	*12928.17*	**12927.96**	2.06	0.96
I-300	11990.33	8447.17	47715.46	*11990.33*	11961.57	*11990.33*	11977.55	*11990.33*	11978.78	*11990.33*	**11980.50**	11.00	6.32
I-400	11131.50	7771.50	69799.63	*11131.50*	11130.16	*11131.50*	11130.62	*11131.50*	*11131.50*	*11131.50*	*11131.50*	0	1.22
I-500	10863.33	5870.17	92542.62	*10863.33*	10857.06	*10863.33*	10860.65	*10863.33*	10862.95	*10863.33*	**10863.19**	0.69	4.36
I-600	9751	5424.50	98901.05	*9751*	**9751**	-	-	-	-	*9751*	9750.95	0.46	2.61
I-700	9559.33	9158.27	50336.56	*9559.33*	*9559.33*	-	-	-	-	*9559.33*	*9559.32*	0.13	4.68
I-800	9354.00	4691.17	141644.54	*9354*	9340.83	-	-	-	-	*9354*	**9354**	0	9.46
I-900	9137.17	4830	155975.89	*9137.17*	9129.40	-	-	-	-	*9137.17*	**9137.11**	0.53	22.77
I-1000	9013.17	3856.50	189733.32	*9013.17*	8989.95	-	-	-	-	*9013.17*	**9013.17**	0	13.60
II-500_600	11875.83	7573.33	251142.53	*11875.83*	11845.53	11774.33	11735.08	11811.50	11762.20	*11875.83*	**11875.83**	0	8.77
II-500_800	10998.17	6991.83	320351.81	*10998.17*	10994.63	10911.33	10833.58	10918.17	10876.95	*10998.17*	**10998.17**	0	1.81
II-500_1000	10135	8397.67	190315.42	10036.50	10007	10073.83	9956.68	10062.50	9985.55	**10232.17**	**10232.17**	0	1.47
II-600_500	13121.67	7985.33	156801.39	*13121.67*	13089.42	13039	12983.17	13044.67	13004.45	*13121.67*	**13121.67**	0	16.16
II-600_600	12482.50	7011	158618.84	*12482.50*	12481.50	12372	12282.60	12435	12377.47	*12482.50*	**12482.50**	0	1.84
II-800_500	13854.17	6381.17	161886.86	*13854.17*	13758.88	13741.67	13655.15	13772.83	13689.05	*13854.17*	**13854.17**	0	42.26
II-800_800	11494.67	7848	194335.48	11491.50	11491.50	11420.17	11297.97	11453.83	11408.35	**11576.50**	**11576.32**	0.55	49.89
II-1000_500	14279.17	5618.17	357638.27	14208.17	14118.75	14237.33	14101.93	14252.50	14145.72	**14316.17**	**14313.63**	7.49	106.68
II-1000_1000	10880.50	6016.83	263776.07	10880.50	10810.15	10662.17	10572.15	10712	10670.28	**10947.33**	**10947.33**	0	36.96
#Avg	11358.12	7240.80	154902.15	11349.04	11323.75	-	-	-	-	**11373.01**	**11372.32**	1.21	17.51

the f_{best} values on the remaining instances. The superior $\#Avg$ value of f_{avg} in the last row demonstrates that ANRMA achieves better overall performance on this set of large instances. Moreover, the consistently small std and t_{avg} values indicate that ANRMA achieves high robustness and efficiency across all instance sizes in the three benchmark sets. Overall, these results demonstrate that ANRMA is highly competitive, matching all BKVs and achieving 15 new lower bounds.

To further illustrate this comparison, Figure 1 provides the performance profiles (see [34]) of f_{best} and f_{avg} of the references on the instance groups of Set II and instances of Set III. Given a set $\mathcal{A}$ of references and a set $\mathcal{I}$ of instances, the performance ratio is determined by $r_{a,i} = \frac{max\{f_{a,i}:a\in\mathcal{A}\}}{f_{a,i}}$, where $f_{a,i}$ is the f_{best} or f_{avg} values of instance i attained by the algorithm a. In the figure, $X\text{-}axis$ displays the performance ratio while the $Y\text{-}axis$ displays the percentage of instances solved by each reference. The intersections with the $Y\text{-}axis$ indicates the fractions of each algorithm can attain the best f_{best} or f_{avg} results among all the references. From Figure 1, we can observe that AMRMA consistently outperforms the competitors across Sets II and III for

Table 4
Computational results and comparisons of the ANRMA algorithm with reference algorithms on the 18 SUKP benchmark instance of Set III (large size).

Instance	BKV	CPLEX		I2PLS [12]		ATS-DLA [13]		MSBTS [14]		ANRMA (this work)			
		LB	*UB*	f_{best}	f_{avg}	f_{best}	f_{avg}	f_{best}	f_{avg}	f_{best}	f_{avg}	*std*	t_{avg}
850_1000_0.10_0.75	10028	4906	157885.02	9944	9851.90	*10028*	*10028*	*10028*	*10028*	*10028*	*10028*	0	3.83
850_1000_0.15_0.85	8957	5744	187274.76	8851	8838.30	*8957*	*8957*	*8957*	8956.50	*8957*	*8957*	0	13.75
1000_850_0.10_0.75	10623	5755	175483.17	10464	10434.30	*10623*	10534.40	*10623*	10622.79	*10623*	**10623**	0	5.14
1000_850_0.15_0.85	9565	4325	206890.27	9532	9394.70	9498	9469.80	*9565*	9554.85	*9565*	**9565**	0	24.77
1000_1000_0.10_0.75	10771	5157	197239.72	10723	10515.40	*10771*	*10771*	*10771*	*10771*	*10771*	*10771*	0	2.88
1000_1000_0.15_0.85	9368	4006	216218.67	9342	9235	*9368*	*9368*	*9368*	9350.44	*9368*	*9368*	0	47.65
2850_3000_0.10_0.75	9260	0	582452.05	8206	8206	*9260*	*9260*	8709	8558.10	*9260*	*9260*	0	69.10
2850_3000_0.15_0.85	8066	0	670002.75	7132	7132	*8066*	**8066**	7639	7395.64	*8066*	8061.68	6.17	456.95
3000_2850_0.10_0.75	9095	5602	619565.48	8192	8191.90	9095	8990.80	8665	8343.36	**9097**	**9095.02**	0.20	688.60
3000_2850_0.15_0.85	8151	0	707055.56	7132	7131.80	8151	8114.20	7632	7492	**8282**	**8282**	0	52.08
3000_3000_0.10_0.75	9175	4110	628923.41	8206	8204.70	*9175*	**9175**	8703	8360.07	*9175*	9164.78	6.22	512.14
3000_3000_0.15_0.85	8064	0	702329.58	7634	7632	*8064*	**8064**	7634	7389.68	*8064*	8063.27	3.68	346.37
4850_5000_0.10_0.75	9480	0	1014873.84	9199	9199	*9480*	*9480*	8744	8707.60	*9480*	*9480*	0	190.11
4850_5000_0.15_0.85	8226	0	1158975.82	7676	7676	*8226*	**8193.20**	7676	7649.30	*8226*	8127.58	14.67	558.85
5000_4850_0.10_0.75	9207	0	1035674.22	8731	8730.70	*9207*	9152.60	8731	8614.39	*9207*	**9155.72**	24.66	657.05
5000_4850_0.15_0.85	8085	0	1189791.22	7653	7648.70	*8085*	8077.40	7653	7559.94	*8085*	**8081.62**	3.59	474.52
5000_5000_0.10_0.75	9092	0	1025031.96	8226	8226	*9092*	**9092**	8720	8448.30	*9092*	9080.07	10.59	438.35
5000_5000_0.15_0.85	8090	0	1192469.91	7660	7660	*8090*	8088	7660	7591.88	*8090*	**8090**	0	171.25
#Avg	9072.39	2200	648229.86	8583.50	8550.47	9068.67	9048.97	8748.78	8632.99	**9079.78**	**9069.65**	3.88	261.85

both f_{best} and f_{avg}, as its profile rises quickly and stays above the others across the entire range of performance ratios. In contrast, the reference methods' curves lie lower and farther to the right, indicating smaller win rates at tight ratios and requiring larger ratios to achieve performance comparable to AMRMA.

4.4 Solving the BMCP using the ANRMA algorithm

To demonstrate the generality of the proposed ANRMA algorithm, we apply it to the BMCP [7,8], which is closely related to the SUKP as discussed in Section 1. We keep the overall structure and parameter settings of the ANRMA algorithm unchanged, and only modify the data structure and the computation of the dynamic profit-to-weight ratio to fit the BMCP. The experiments are carried out on two sets of BMCP benchmark instances ranging from 585 to 5000 items and elements: (i) Set I from [7], consisting of 30 medium-sized instances with 585 to 1000 items and elements; (ii) Set II from [35], comprising 60 large instances with 1100 to 5000 items and elements. For ease of presentation, the BMCP instances are compressed using the same aggregation method as for the SUKP instances (see Section

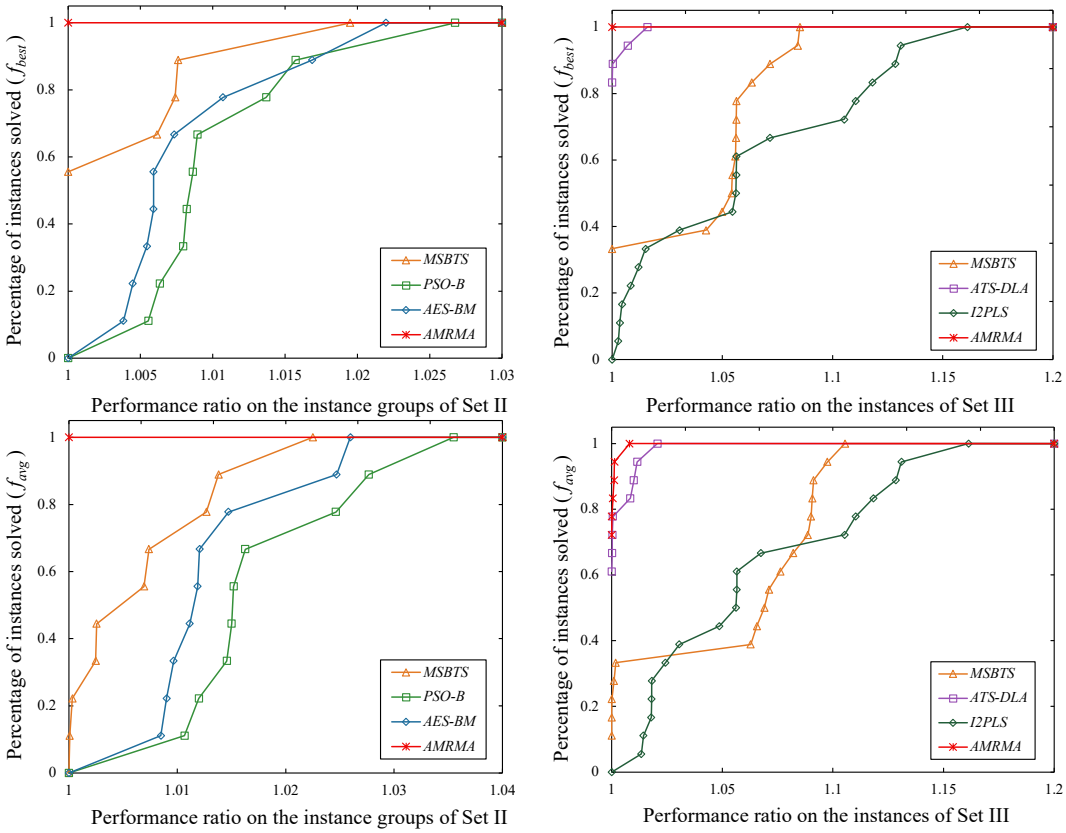


Fig. 1. Performance profiles of the references on f_{bset} and f_{avg} on instance groups of Set II (left) and the instances of Set III (right).

4.1).

For comparison purposes, we adopt three state-of-the-art algorithms from the literature as reference methods: probability learning based tabu search (PLTS) [7], iterated hyperplane search (IHS) [8], and variable-depth local search (VDLS) [35]. Following these references, each instance is solved independently 30 times with a cutoff times of 600 seconds for Set I and 1800 seconds for Set II. Since IHS doesn't report results for Set II in [8], we run IHS together with ANRMA under the same conditions.

The aggregated results of the references on Sets I and II of the BMCP are presented in Table 5. The detailed results for each BMCP instance are given in Table C.1 and Table C.2 of Appendix C. The second column in all these tables represents the best-known values (BKV) reported in the literature. From Table 5 and C.1, we can observe that ANRMA matches all BKV results on Set I and achieves four better f_{avg} values, while matching the remaining instances with low time cost, indicating its robustness in solving the BMCP. From the 60 results of Set II in Tables 5 and C.2, we can observe that ANRMA achieves 27 improved best results (new lower bounds) compared with the BKV and matches 32 f_{best} values. Moreover, our algorithm exhibits high robustness, obtaining better f_{avg} values for all

Table 5
Computational results of the ANRMA algorithm and reference algorithms on the BMCP benchmark instances of Set I and Set II (small to medium size, 90 instances aggregated into 15 groups, with each row representing the average over six instances).

Instance group	BKV	PLTS [7]		VDLS [35]		IHS [8]		ANRMA (this work)			
		f_{best}	f_{avg}	f_{best}	f_{avg}	f_{best}	f_{avg}	f_{best}	f_{avg}	std	t_{avg}
BMCP-I-600	70165.50	70081.17	69971.27	*70165.50*	69964.48	*70165.50*	70164.71	*70165.50*	**70165.50**	0	0.92
BMCP-I-700	80745.00	80618.17	80446.35	*80745.00*	80418.55	*80745.00*	80726.61	*80745.00*	**80745.00**	0	1.06
BMCP-I-800	92790.83	92390.00	92251.65	*92790.83*	92592.35	*92790.83*	*92790.83*	*92790.83*	*92790.83*	0	1.71
BMCP-I-900	103514.00	103084.33	102690.34	*103514.00*	103202.95	*103514.00*	103459.46	*103514.00*	**103514.00**	0	1.45
BMCP-I-1000	114237.17	113615.50	112827.74	*114237.17*	113726.07	*114237.17*	114234.70	*114237.17*	**114237.17**	0	1.34
BMCP-II-1100	148369.33	147059.17	146529.90	148369.33	147741.23	*148425.00*	148420.24	*148425.00*	**148425.00**	0.00	3.04
BMCP-II-1200	161970.00	159992.00	159097.68	*161970.00*	160864.48	*161970.00*	161959.24	*161970.00*	**161964.32**	30.58	44.21
BMCP-II-1300	173686.83	171967.33	170890.85	173686.83	173039.73	*173766.50*	173715.34	*173766.50*	**173766.50**	0.00	47.14
BMCP-II-1400	186926.67	185629.00	183836.35	186926.67	186204.25	*187044.00*	187018.26	*187044.00*	**187044.00**	0.00	27.66
BMCP-II-1500	199291.50	197710.67	195625.58	199291.50	198214.05	**199444.50**	199397.53	199419.17	**199417.17**	10.77	87.54
BMCP-II-4200	1873077.17	1818227.83	1791447.07	1873077.17	1865469.98	1871029.83	1871029.83	**1873946.83**	**1873200.21**	432.14	530.69
BMCP-II-4400	1958449.67	1884967.17	1865361.22	1958449.67	1950326.70	*1959748.33*	**1959748.33**	*1959748.33*	1958445.98	597.41	713.26
BMCP-II-4600	2044525.00	1972251.83	1949389.50	2044525.00	2036719.28	2045370.83	2045370.83	**2046640.17**	**2045442.11**	799.76	632.97
BMCP-II-4800	2127483.00	2053907.00	2022380.05	2127483.00	2117718.27	2129088.83	2128314.93	**2131008.33**	**2128809.85**	1457.63	537.34
BMCP-II-5000	2206600.17	2109371.00	2089748.80	2206600.17	2196549.38	2205665.67	2205566.78	**2208020.50**	**2206579.68**	1056.86	823.88
#Avg	769455.46	744058.14	735499.62	769455.46	766183.45	769533.73	769461.17	**770096.09**	**769636.49**	292.34	230.28

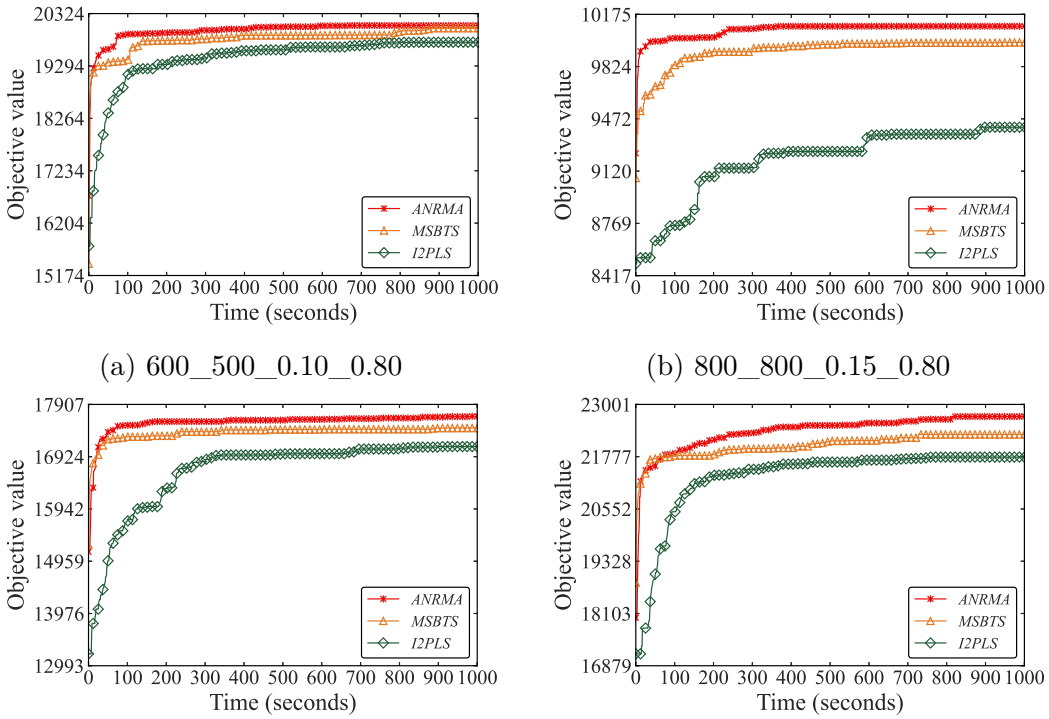
instances with only one exception, where the result is the same. When compared with IHS, a dedicated algorithm for the BMCP, our ANRMA algorithm remains highly competitive, achieving 14 better f_{best} results and 21 better f_{avg} results, while matching the results on most of the remaining instances. These outcomes demonstrate the generality of our algorithm, which can effectively and efficiently solve SUKP-related problems such as the BMCP without requiring problem-specific modifications, highlighting its potential as a practical tool for real-world problems modeled under similar constraints.

5 Analysis

In this section, we study the influences of the main components of the ANRMA algorithm, including the convergence analysis (see Section 5.1), the ANR strategy employed in the SBTS procedure (see Section 3.3.1) and the diversity-driven greedy crossover (see Section 3.4). These experiments are conducted on instances with more than 500 items to better examine the performance of the ANRMA algorithm on medium and large instances. To facilitate the presentation of results, we adopt the instance aggregation method introduced in Section 4.1, where each group consists of six instances with identical item and element sizes.

5.1 Convergence analysis

The ANRMA algorithm demonstrates high efficiency by finding high-quality solutions in a remarkably short time. To assess its convergence performance, we compare ANRMA with two reference algorithms, I2PLS and MSBTS, whose codes are publicly available online. We select four representative medium and large instances from Sets II and III. Figure 2 shows the convergence plots of the compared algorithms, where each data point represents the mean value of the f_{best} at the corresponding time points across 10 independent runs.



(a) 600_500_0.10_0.80 (b) 800_800_0.15_0.80

(c) 1000_500_0.10_0.75 (d) 1000_500_0.10_0.80

Fig. 2. Convergence analysis of ANRMA and reference algorithms.

We observe that ANRMA consistently outperforms MSBTS and I2PLS across all tested instances, achieving higher solution quality with a faster convergence rate. MSBTS can obtain better solutions at the beginning of some large instances (1000_500_0.10_0.75 and 1000_500_0.10_0.80). However, its convergence is basically slower than that of ANRMA and fails to achieve solutions of higher qualities. I2PLS ranks last, as it encounters challenges in converging for large instances.

5.2 Analysis of adaptive neighborhood reduction strategy

The adaptive neighborhood reduction (ANR) strategy aims to effectively reduce the neighborhood size and make the search on promising items. To evaluate the influence of the ANR strategy on the algorithm performance, we construct two variants of the ANRMA algorithm. (i) MA denotes the memetic algorithm that explores the entire neighborhood induced by the *swap* operator, without applying the ANR strategy. (ii) NRMA denotes the neighborhood reduction-based memetic algorithm in which the reduced neighborhood is obtained solely according to the dynamic profit-to-weight ratio with respect to the current solution, without employing the dynamic scoring mechanism (see Section 3.3.1). In particular, this means that lines 6–11 of Algorithm 4 are disabled, and the top n_a items with the highest R_D^+ values are directly selected in a single step to construct the reduced neighborhood $N_r(S)$.

Table 6 presents the comparative results of MA, NRMA and ANRMA. The first column describes the attributes of instance groups, which comprise the instance set, item size n and element size m formatted as *set_n_m*. The results of the best objective (f_{best}) and the average objective (f_{avg}) of each algorithm are respectively reported in the remaining columns. We also report the average runtime t_{avg} (in seconds) to further examine the impact of the ANR strategy on computational time. Row *#Avg* shows the average value of the corresponding column, while row *#Wins*, *#Ties* and *#Losses* respectively count the better, equal and worse result of f_{best} or f_{avg} obtained by the corresponding variant compared to ANRMA. The results of the Wilcoxon signed-rank test are reported in the row *p-value* at the bottom.

Table 6: Comparison of MA (without the ANR strategy), NRMA (without the dynamic scoring mechanism) and ANRMA (with the ANR strategy and the dynamic scoring mechanism).

Instance group	MA			NRMA			ANRMA		
	f_{best}	f_{avg}	t_{avg}	f_{best}	f_{avg}	t_{avg}	f_{best}	f_{avg}	t_{avg}
I_500_485	10983.50	10953.91	242.47	*11004.50*	11001.04	16	*11004.50*	**11004.08**	9.78
I_500_500	*10815*	10789.32	137.12	*10815*	*10815*	1.92	*10815*	*10815*	2.02
I_585_600	*9824.50*	*9824.50*	66.47	*9824.50*	*9824.50*	1.08	*9824.50*	*9824.50*	1.29
I_600_585	*9635.50*	9629.03	235.42	*9635.50*	9635.08	2.42	*9635.50*	**9635.50**	2.93
I_600_600	*9793*	9792.66	124.98	*9793*	9792.30	10.08	*9793*	**9792.86**	3.60
I_685_700	*9648.50*	9615.22	393.44	*9648.50*	*9648.50*	2.93	*9648.50*	*9648.50*	4.63
I_700_685	*9522*	9498.30	255.95	*9522*	*9522*	3.42	*9522*	*9522*	3.16
I_700_700	9481.50	9481.50	95.26	*9507.50*	**9507.50**	5.99	*9507.50*	9507.46	6.24
I_785_800	9023.50	9006.07	362.61	*9065*	9064.59	16.43	*9065*	**9065**	19.98

Continued on next page

Table 6 – continued from previous page

Instance group	MA			NRMA			ANRMA		
	f_{best}	f_{avg}	t_{avg}	f_{best}	f_{avg}	t_{avg}	f_{best}	f_{avg}	t_{avg}
I_800_785	*9480.50*	9357.09	692.62	*9480.50*	*9480.50*	4.15	*9480.50*	*9480.50*	5.84
I_800_800	9446.50	9384.71	380.98	*9516.50*	*9516.50*	1.12	*9516.50*	*9516.50*	2.55
I_885_900	*8871.50*	8813.31	552.63	*8871.50*	*8871.50*	9.15	*8871.50*	*8871.50*	16.87
I_900_885	*9172.50*	9055.16	85.84	*9172.50*	*9172.34*	15.69	*9172.50*	*9172.34*	47.91
I_900_900	9325	9302.87	486.82	*9367.50*	*9367.50*	1.97	*9367.50*	*9367.50*	3.53
I_985_1000	8870	8836.50	246.74	*8923*	*8923*	14.66	*8923*	*8923*	13.90
I_1000_985	9068.50	9010.94	212.27	*9072*	*9072*	10.64	*9072*	*9072*	22.95
I_1000_1000	9033.50	8915.97	612.23	*9044.50*	*9044.50*	3.58	*9044.50*	*9044.50*	3.96
II_500_600	11803	11733.15	307.33	*11875.83*	11872.93	5.50	*11875.83*	**11875.83**	8.77
II_500_800	10914	10828.87	138.39	*10998.17*	*10998.17*	1.66	*10998.17*	*10998.17*	1.81
II_500_1000	10157.17	10099.88	178.47	*10232.17*	*10232.17*	1.93	*10232.17*	*10232.17*	1.47
II_600_500	13097.33	13043.25	371.29	*13121.67*	13114.32	13.08	*13121.67*	**13121.67**	16.16
II_600_600	12372	12351.72	474.53	*12482.50*	*12482.50*	1.99	*12482.50*	*12482.50*	1.84
II_800_500	13831.50	13733.47	521.56	*13854.17*	13852.27	41.41	*13854.17*	**13854.17**	42.26
II_800_800	11487.83	11420.37	252.31	*11576.50*	11576.25	22.82	*11576.50*	**11576.32**	49.89
II_1000_500	14272.17	14144.73	537.99	14296.33	14279.98	144.74	**14316.17**	**14313.63**	106.68
II_1000_1000	10798.67	10766.03	409.02	*10947.33*	*10947.33*	18.95	*10947.33*	*10947.33*	36.96
III_850_1000	9465	9433.39	451.85	*9492.50*	*9492.50*	4.75	*9492.50*	*9492.50*	8.79
III_1000_850	*10094*	10044.49	855.13	*10094*	*10094*	12.68	*10094*	*10094*	14.95
III_1000_1000	10002	9951.06	196.69	*10069.50*	*10069.50*	5.18	*10069.50*	*10069.50*	25.26
III_2850_3000	8610	8604.33	251.83	*8663*	**8661.28**	213.87	*8663*	8660.84	263.02
III_3000_2850	8542	8502.16	315.39	*8689.50*	8654.54	469.50	*8689.50*	**8688.51**	370.34
III_3000_3000	8565	8528.43	414.99	*8619.50*	8596.24	574.79	*8619.50*	**8614.03**	429.25
III_4850_5000	8662.50	8661.83	813.64	*8853*	**8813.46**	519.57	*8853*	8803.79	374.48
III_5000_4850	8612	8506.69	404.20	*8646*	**8620.24**	357	*8646*	8618.67	565.79
III_5000_5000	*8591*	8417.09	465.84	*8591*	8582.78	297.01	*8591*	**8585.04**	304.80
#Avg	10053.49	10001.08	358.41	10096.18	10091.39	80.79	**10096.74**	**10094.04**	79.82
#Wins	0	0	-	0	4	-	-	-	-
#Ties	11	1	-	30	17	-	-	-	-
#Losses	24	34	-	5	14	-	-	-	-
p-value	1.82e-05	3.65e-07	-	1.80e-01	2.68e-02	-	-	-	-

We can observe from Table 6 that compared to ANRMA, MA fails to attain 24 average f_{best} results and 34 f_{avg} results out of 35 instance groups. The small *p-values* (< 0.05) from the Wilcoxon signed-rank test also confirm the significant difference on the f_{best} and f_{avg} results between MA and ANRMA, indicating that the ANR is a crucial strategy to the algorithm. Compared with the NRMA variant, the ANRMA algorithm achieves 5 better f_{best} results and 14 better f_{avg} results, indicating that the dynamic scoring mechanism contributes positively to improving the robustness of the algorithm. Moreover,

according to the t_{avg} values, ANRMA reaches its f_{best} values substantially faster than MA, further confirming the effectiveness of the ANR strategy in improving search efficiency.

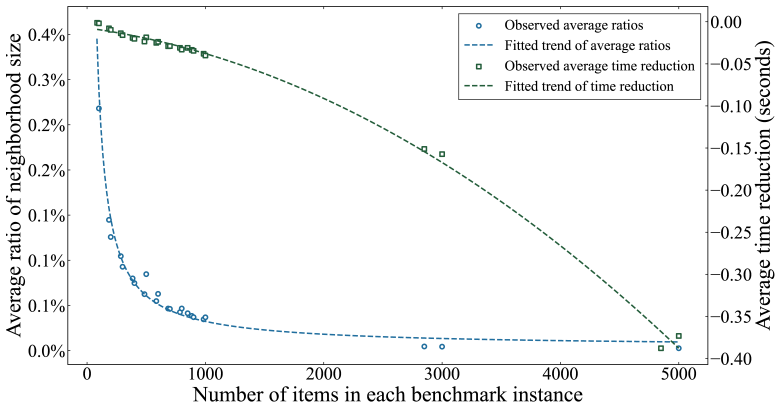


Fig. 3. Comparison of neighborhood sizes and time reduction with and without the ANR strategy.

We further illustrate the effect of the ANR strategy on neighborhood size and exploration time in Figure 3. Specifically, we test MA (without ANR) and ANRMA (with ANR) on all 132 benchmark instances and record, for each instance, the neighborhood size and the time required to explore one neighborhood. Let N and N_r denote the sizes of the full and reduced neighborhoods, we then compute the ratio of neighborhood size $R_{size} = \frac{|N_r|}{|N|} \times 100\%$. We also record the time required to explore the full and reduced neighborhoods, denoted by t_N and t_{N_r} respectively, and compute the time reduction as $\Delta_t = t_{N_r} - t_N$. The $X\text{-}axis$ lists the benchmark instances in ascending order of size, and instances with similar sizes are grouped so that each plotted point represents the average value within its group. The left $Y\text{-}axis$ corresponds to R_{size} and the right $Y\text{-}axis$ corresponds to Δ_t. The scattered points show the observed average values of both R_{size} and Δ_t, while the dashed curves represent their fitted trends.

From Figure 3, we can observe that as the instance size increases, both R_{size} and Δ_t exhibit a clear decreasing trend, indicating that the relative reduction of the neighborhood becomes stronger and the time to explore one neighborhood is substantially reduced. This confirms that the ANR strategy significantly improves search efficiency for larger instances. This finding is consistent with the computational results in Table 6, which show that ANRMA outperforms the MA variant.

5.3 Analysis of crossover operator

ANRMA uses the crossover DGC for the SUKP based on the dynamic profit-to-weight ratio introduced in Section 3.3.2. To analyze the influence of the DGC crossover on the algorithm performance, we construct three variants of the ANRMA algorithm: (i) ANRMA_RC denotes ANRMA with a random crossover, where each item selected by at least one parent (i.e., chosen by one or both parents) has a 0.5 probability of being added to the offspring solution, subject to the knapsack capacity constraint, while items not selected by either parent are excluded. (ii) ANRMA_BC denotes the ANRMA that adopts the backbone-based crossover ([32,31]) where all items simultaneously selected by both parent solutions are retained and items selected by only one of the parents are randomly selected within the remaining knapsack capacity. (iii) ANRMA_S-GROA denotes the ANRMA that employs the popular static profit-to-weight ratio S-GROA used in [11,15,22,24] in Steps 2 and 3 of Algorithm 5 without the dynamic profit-to-weight ratio.

Table 7: Comparison of ANRMA_RC (with the random crossover), ANRMA_BC (with the backbone-based crossover), ANRMA_S-GROA (with the S-GROA) and ANRMA (with the DGC and the dynamic profit-to-weight ratio).

Instance group	ANRMA_RC		ANRMA_BC		ANRMA_S-GROA		ANRMA	
	f_{best}	f_{avg}	f_{best}	f_{avg}	f_{best}	f_{avg}	f_{best}	f_{avg}
I_500_485	*11004.50*	10997.05	10994	10960.69	*11004.50*	11003.69	*11004.50*	**11004.08**
I_500_500	*10815*	10814.79	*10815*	10721.33	*10815*	*10815*	*10815*	*10815*
I_585_600	*9824.50*	9823.99	*9824.50*	9770.81	*9824.50*	*9824.50*	*9824.50*	*9824.50*
I_600_585	*9635.50*	9635.29	*9635.50*	9595.18	*9635.50*	*9635.50*	*9635.50*	*9635.50*
I_600_600	*9793*	**9793**	*9793*	9776.88	*9793.00*	9792.86	*9793*	9792.86
I_685_700	*9648.50*	*9649*	*9648.50*	9641.28	*9648.50*	*9648.50*	*9648.50*	*9648.50*
I_700_685	*9522*	*9522*	*9522*	9504.50	*9522*	*9522*	*9522*	*9522*
I_700_700	*9507.50*	9507.46	*9507.50*	9476.36	*9507.50*	*9507.50*	*9507.50*	9507.46
I_785_800	*9065*	9064.17	*9065*	9016.80	*9065*	9064.17	*9065*	**9065**
I_800_785	*9480.50*	9480.00	*9480.50*	9404.22	*9480.50*	*9480.50*	*9480.50*	*9480.50*
I_800_800	*9516.50*	*9516.50*	*9516.50*	9493.65	*9516.50*	*9516.50*	*9516.50*	*9516.50*
I_885_900	*8871.50*	*8871.50*	*8871.50*	8853.54	*8871.50*	8871.42	*8871.50*	*8871.50*
I_900_885	*9172.50*	9170.74	9156.50	9142.68	*9172.50*	*9172.34*	*9172.50*	*9172.34*
I_900_900	*9367.50*	*9367.50*	*9367.50*	9346.58	*9367.50*	*9367.50*	*9367.50*	*9367.50*
I_985_1000	*8923*	*8923*	*8923*	8883.78	*8923*	8922.68	*8923*	*8923*
I_1000_985	*9072*	9071.90	*9072*	9051.85	*9072*	*9072*	*9072*	*9072*
I_1000_1000	*9044.50*	*9044.50*	*9044.50*	9023.29	*9044.50*	*9044.50*	*9044.50*	*9044.50*
II_500_600	*11875.83*	11871.48	11861.33	11807.98	*11875.83*	*11875.83*	*11875.83*	*11875.83*

Continued on next page

Table 7 – continued from previous page

Instance group	ANRMA_RC		ANRMA_BC		ANRMA_S-GROA		ANRMA	
	f_{best}	f_{avg}	f_{best}	f_{avg}	f_{best}	f_{avg}	f_{best}	f_{avg}
II_500_800	*10998.17*	*10998.17*	*10998.17*	10977.38	*10998.17*	*10998.17*	*10998.17*	*10998.17*
II_500_1000	*10232.17*	*10232.17*	*10232.17*	10214.75	*10232.17*	*10232.17*	*10232.17*	*10232.17*
II_600_500	*13121.67*	13107.15	13114.17	13018.88	*13121.67*	13117.82	*13121.67*	**13121.67**
II_600_600	*12482.50*	*12482.50*	*12482.50*	12463.03	*12482.50*	*12482.50*	*12482.50*	*12482.50*
II_800_500	*13854.17*	13829.50	13832.67	13689.92	*13854.17*	13850.10	*13854.17*	**13854.17**
II_800_800	*11576.50*	*11576.32*	*11576.50*	11544.05	*11576.50*	11575.95	*11576.50*	*11576.32*
II_1000_500	14284.50	14250.43	14280.33	14185.98	*14316.17*	14275.72	*14316.17*	**14313.63**
II_1000_1000	*10947.33*	*10947.33*	*10947.33*	10916.73	*10947.33*	*10947.33*	*10947.33*	*10947.33*
III_850_1000	*9492.50*	*9492.50*	*9492.50*	9485.42	*9492.50*	*9492.50*	*9492.50*	*9492.50*
III_1000_850	*10094*	*10094*	*10094*	10071.55	*10094*	*10094*	*10094*	*10094*
III_1000_1000	*10069.50*	*10069.50*	*10069.50*	10067.55	*10069.50*	*10069.50*	*10069.50*	*10069.50*
III_2850_3000	8660.50	8656.46	8644.50	8580.49	*8663*	**8661.81**	*8663*	8660.84
III_3000_2850	8688.50	8641.92	8647	8522.27	8688.50	8665.90	**8689.50**	**8688.51**
III_3000_3000	8612.50	8581.35	8593.50	8554.97	*8619.50*	8599.54	*8619.50*	**8614.03**
III_4850_5000	8803	8801.59	8802.50	8794.17	8803	8801.57	**8853**	**8803.79**
III_5000_4850	8617	8613.47	8617	8602.88	*8646*	8612.08	*8646*	**8618.67**
III_5000_5000	*8591*	8584.80	*8591*	8573.80	*8591*	8583.08	*8591*	**8585.04**
#Avg	10096.40	10091.54	10088.96	10049.58	10095.29	10091.33	**10096.74**	**10094.04**
#Wins	0	1	0	0	0	2	-	-
#Ties	25	13	22	0	28	18	-	-
#Losses	10	21	13	35	7	15	-	-
p-value	2.77e-02	1.82e-04	3.35e-03	2.48e-07	1.80e-01	3.14e-03	-	-

Table 7 presents the computational results of the compared variants, with the same row and column definitions as those in Table 6. Compared to ANRMA, ANRMA_RC and ANRMA_BC respectively fail 6 and 11 f_{best} results. In terms of the f_{avg}, ANRMA_RC also fails 21 results while ANRMA_BC is dominated in all 35 results compared to the ANRMA. The small *p-values* (< 0.05) confirm the significant differences among the results, indicating the obvious advantages of the DGC. Besides, although ANRMA_S-GROA only loses 7 f_{best} results and reaches the same f_{best} results on the rest of the instances, it still fails 15 f_{avg} results compared to the ANRMA with a significant difference confirmed by the small *p-value* (3.14e-3), indicating the advantage of the dynamic profit-to-weight ratio.

6 Conclusion

The set-union knapsack problem SUKP continues to draw significant interest due to its NP-hard complexity and broad applicability in practical settings. In this work, we introduced ANRMA, a new adaptive neighborhood reduction-based memetic algorithm for better solving the problem. The proposed approach integrates a solution-based tabu search enhanced by a dynamic scoring based adaptive neighborhood reduction strategy to effectively locate high-quality local optima. Furthermore, a specially designed diversity-driven greedy crossover operator and an adaptive population updating rule are incorporated to promote exploration by enhancing the population diversity.

Computational experiments are conducted on three sets of 132 benchmark instances of the SUKP, with instance sizes ranging from 85 to 5000. Compared to state-of-the-art SUKP algorithms, the proposed ANRMA algorithm demonstrates strong competitiveness in terms of solution quality, robustness, and convergence speed. Specifically, our algorithm achieved 12 new best results on medium instances with 500 to 1000 items and elements and 2 new best results on large instances with 3000 to 5000 items and elements. The proposed algorithm features an adaptive mechanism that dynamically adjusts its search strategy based on the instance size, while requiring no parameter to be specifically tuned. This makes it a valuable addition to the toolbox for solving the SUKP and enables it to effectively accommodate the varying instance scales encountered in real-world applications. Extensive experiments on 90 benchmark instances of the BMCP, a problem closely related to the SUKP, show that our algorithm achieves 27 improved best results (new lower bounds) while matching the results on most of the remaining instances, further demonstrating its competitiveness.

This work can be further improved from two perspectives. First, although numerous algorithms have been proposed for the SUKP, the development of powerful exact methods remains a valuable research direction for determining optimal solutions or tighter bounds. Second, beyond the BMCP, the proposed algorithm can be extended to solve other SUKP-related problems, such as the set covering problem [36] due to its generality.

Acknowledgment

We are grateful to the reviewers for their useful comments and suggestions which helped us to significantly improve the paper. This work was partially

supported by the National Natural Science Foundation of China (No. 72301036, 72401088) and the Jiangsu Natural Science Foundation (No. BK20241504). Support from the High-performance Computing Platform of BUPT is also acknowledged.

References

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

[2] O. Goldschmidt, D. Nehme, G. Yu, Note: On the set-union knapsack problem, Naval Research Logistics 41 (6) (1994) 833–842.

[3] B. Schneier, Applied cryptography: Protocols, algorithms, and source code in C, John wiley & Sons, 2007.

[4] M. Tu, L. Xiao, System resilience enhancement through modularization for large scale cyber systems, in: 2016 IEEE/CIC International Conference on Communications in China, IEEE, 2016, pp. 1–6.

[5] Z. Wei, J.-K. Hao, Kernel based tabu search for the set-union knapsack problem, Expert Systems with Applications 165 (2021) 113802.

[6] E. Chlamtác, M. Dinitz, C. Konrad, G. Kortsarz, G. Rabanca, The densest k-subhypergraph problem, SIAM Journal on Discrete Mathematics 32 (2) (2018) 1458–1477.

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

[8] Z. Wei, J.-K. Hao, Iterated hyperplane search for the budgeted maximum coverage problem, Expert Systems with Applications 214 (2023) 119078.

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

[10] A. Arulselvan, A note on the set union knapsack problem, Discrete Applied Mathematics 169 (2014) 214–218.

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

[12] Z. Wei, J.-K. Hao, Iterated two-phase local search for the set-union knapsack problem, Future Generation Computer Systems 101 (2019) 1005–1017.

[13] Y. Zhou, M. Zhao, M. Fan, Y. Wang, J. Wang, An efficient local search for large-scale set-union knapsack problem, Data Technologies and Applications 55 (2) (2021) 233–250.

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

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

[16] I. Dahmani, M. Ferroum, M. Hifi, Effect of backtracking strategy in population-based approach: the case of the set-union knapsack problem, Cybernetics and Systems 53 (1) (2022) 168–185.

[17] J. Zhao, M. Hifi, An adaptive evolutionary search-based method for efficiently tackling the set-union knapsack problem, Information Sciences (2024) 120855.

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

[19] Y. He, X. Wang, Group theory-based optimization algorithm for solving knapsack problems, Knowledge-Based Systems 219 (2021) 104445.

[20] I. Dahmani, M. Ferroum, M. Hifi, An iterative rounding strategy-based algorithm for the set-union knapsack problem, Soft Computing 25 (21) (2021) 13617–13639.

[21] R. Durgut, M. E. Aydin, I. Atli, Adaptive operator selection with reinforcement learning, Information Sciences 581 (2021) 773–790.

[22] F. B. Ozsoydan, İ. Gölcük, A reinforcement learning based computational intelligence approach for binary optimization problems: The case of the set-union knapsack problem, Engineering Applications of Artificial Intelligence 118 (2023) 105688.

[23] J. Zhao, M. Hifi, T. Saadi, A hybrid machine learning method for solving the set union knapsack problem, in: International Conference on Intelligent Systems Design and Applications, Springer, 2023, pp. 299–309.

[24] E. Sonuç, E. Özcan, Cuda-based parallel local search for the set-union knapsack problem, Knowledge-Based Systems (2024) 112095.

[25] I. Dahmani, M. Ferroum, M. Hifi, S. Sadeghsa, A hybrid swarm optimization-based algorithm for the set-union knapsack problem, in: 2020 7th international conference on control, decision and information technologies, Vol. 1, IEEE, 2020, pp. 1162–1167.

[26] I. Dahmani, M. Ferroum, M. Hifi, The local branching as a learning strategy in the evolutionary algorithm: The case of the set-union knapsack problem, Journal of Advances in Information Technology 13 (3) (2022).

[27] I. M. Coelho, S. Hanafi, R. Todosijevic, M. Ratli, B. Gendron, Variable neighborhood search with dynamic exploration for the set union knapsack problem, in: Combinatorial Optimization and Applications: A Tribute to Bernard Gendron, Springer, 2024, pp. 17–35.

[28] J.-K. Hao, Memetic algorithms in discrete optimization, in: Handbook of Memetic Algorithms, Springer, 2012, pp. 73–94.

[29] Z. Ou, S. Wang, Finding robust and influential nodes on directed networks using a memetic algorithm, Swarm and Evolutionary Computation 87 (2024) 101542.

[30] F. Glover, M. Laguna, Tabu search, Springer, 1998.

[31] Z. Wei, J.-K. Hao, A threshold search based memetic algorithm for the disjunctively constrained knapsack problem, Computers & Operations Research 136 (2021) 105447.

[32] Y. Zhou, J.-K. Hao, F. Glover, Memetic search for identifying critical nodes in sparse graphs, IEEE transactions on cybernetics 49 (10) (2018) 3699–3712.

[33] X. Lai, J.-K. Hao, F. Glover, Z. Lü, A two-phase tabu-evolutionary algorithm for the 0–1 multidimensional knapsack problem, Information Sciences 436 (2018) 282–301.

[34] E. D. Dolan, J. J. Moré, Benchmarking optimization software with performance profiles, Mathematical programming 91 (2) (2002) 201–213.

[35] J. Zhou, J. Zheng, K. He, Effective variable depth local search for the budgeted maximum coverage problem, International Journal of Computational Intelligence Systems 15 (1) (2022) 43.

[36] C. Luo, W. Xing, S. Cai, C. Hu, Nusc: an effective local search algorithm for solving the set covering problem, IEEE Transactions on Cybernetics 54 (3) (2022) 1403–1416.

A Computational results on the SUKP instances of Sets I and II

Table A.1 shows the original computational results of the proposed ANRMA algorithm and its comparisons with the references on the 60 small to medium benchmark instances of Set I. The indicators of each column remain the same as those described in Section 4.3. Instances for which the optimal solution is obtained by the CPLEX solver are denoted with an asterisk (*). The best f_{best} and f_{avg} values in each row are highlighted in bold, while equal best results are indicated in *italic*.

Table A.1: Computational results of the ANRMA algorithm and reference algorithms on each of the 60 instances of the SUKP Set I.

Instance	BKV	CPLEX		MSBTS [14]		PSO-B [16]		AES-BM [17]		ANRMA (this work)			
		LB	UB	f_{best}	f_{avg}	f_{best}	f_{avg}	f_{best}	f_{avg}	f_{best}	f_{avg}	std	t_{avg}
100_85_0.10_0.75*	13283	13283	13283.00	*13283*	*13283*	*13283*	*13283*	*13283*	*13283*	*13283*	*13283*	0	0.67
100_85_0.15_0.85*	12479	12479	12479.00	*12479*	12426.53	*12479*	*12479*	*12479*	*12479*	*12479*	*12479*	0	1.23
200_185_0.10_0.75	13521	11205	27506.43	*13521*	*13521*	*13521*	*13521*	*13521*	*13521*	*13521*	*13521*	0	0.58
200_185_0.15_0.85	14215	11625	31365.97	*14215*	13979.58	*14215*	14180.40	*14215*	14177	*14215*	*14215*	0	1.86
300_285_0.10_0.75	11563	9263	44984.89	*11563*	*11563*	*11563*	*11563*	*11563*	*11563*	*11563*	*11563*	0	0.94
300_285_0.15_0.85	12607	6976	50940.43	*12607*	12440.82	*12607*	12542.70	*12607*	12537.70	*12607*	**12549.64**	54.40	28.59
400_385_0.10_0.75	11484	8893	68675.16	*11484*	*11484*	*11484*	*11484*	*11484*	*11484*	*11484*	*11484*	0	0.58
400_385_0.15_0.85	11209	7019	75115.84	*11209*	*11209*	*11209*	11206.60	*11209*	*11209*	*11209*	*11209*	0	1.00
500_485_0.10_0.75	11771	7993	85125.43	*11771*	*11771*	*11771*	11767.90	*11771*	*11771*	*11771*	*11771*	0	1.00
500_485_0.15_0.85	10238	5515	98367.13	*10238*	10210.60	*10238*	10234.60	*10238*	10235.70	*10238*	**10237.16**	4.12	18.56
100_100_0.10_0.75*	14044	14044	14044.00	*14044*	*14044*	*14044*	*14044*	*14044*	*14044*	*14044*	*14044*	0	0.63
100_100_0.15_0.85*	13508	13508	13508.00	*13508*	*13508*	*13508*	*13508*	*13508*	*13508*	*13508*	*13508*	0	0.86
200_200_0.10_0.75	12522	11187	29394.32	*12522*	12520.76	*12522*	*12522*	*12522*	*12522*	*12522*	12520.76	12.34	0.87
200_200_0.15_0.85	12317	9598	31764.16	*12317*	12316.21	*12317*	12303.30	*12317*	*12317*	*12317*	*12317*	0	0.92
300_300_0.10_0.75	12817	10567	45839.91	*12817*	12814.03	*12817*	*12817*	*12817*	*12817*	*12817*	*12817*	0	1.12
300_300_0.15_0.85	11585	6822	48812.36	*11585*	*11585*	*11585*	11572.60	*11585*	*11585*	*11585*	11583.34	11.62	4.23
400_400_0.10_0.75	11665	9405	68734.08	*11665*	11657.30	*11665*	11662.10	*11665*	*11665*	*11665*	*11665*	0	0.82
400_400_0.15_0.85	11325	5911	74045.89	*11325*	*11325*	*11325*	*11325*	*11325*	*11325*	*11325*	*11325*	0	0.89
500_500_0.10_0.75	11249	7500	85184.48	*11249*	*11249*	*11249*	*11249*	*11249*	*11249*	*11249*	*11249*	0	2.03
500_500_0.15_0.85	10381	3948	101964.36	*10381*	10370.98	*10381*	10376.20	*10381*	*10381*	*10381*	*10381*	0	2.00
85_100_0.10_0.75*	12045	12045	12045.00	*12045*	*12045*	*12045*	*12045*	*12045*	*12045*	*12045*	*12045*	0	0.66
85_100_0.15_0.85*	12369	12369	12369.00	*12369*	*12369*	*12369*	*12369*	*12369*	*12369*	*12369*	*12369*	0	0.74
185_200_0.10_0.75	13696	12264	25702.48	*13696*	*13696*	*13696*	*13696*	*13696*	*13696*	*13696*	*13696*	0	0.88
185_200_0.15_0.85	11298	8608	26289.16	*11298*	*11298*	*11298*	*11298*	*11298*	*11298*	*11298*	*11298*	0	0.64
285_300_0.10_0.75	11568	9421	44274.85	*11568*	11567.70	*11568*	*11568*	*11568*	*11568*	*11568*	*11568*	0	0.57
285_300_0.15_0.85	11802	7634	51440.30	*11802*	11798.88	*11802*	*11802*	*11802*	*11802*	*11802*	*11802*	0	2.51
385_400_0.10_0.75	10600	9591	58817.77	*10600*	10599.80	*10600*	*10600*	*10600*	*10600*	*10600*	*10600*	0	2.40
385_400_0.15_0.85	10506	5810	73409.01	*10506*	10505.84	*10506*	*10506*	*10506*	*10506*	*10506*	*10506*	0	1.61
485_500_0.10_0.75	11321	5940	84239.56	*11321*	*11321*	*11321*	11317.40	*11321*	*11321*	*11321*	*11321*	0	1.24
485_500_0.15_0.85	10220	4325	100374.77	*10220*	10219.76	*10220.00*	10218.80	*10220*	*10220*	*10220*	*10220*	0	1.30
600_585_0.10_0.75	9914	7354	96045.09	*9914*	*9914*	-	-	-	-	*9914*	*9914*	0	2.48
600_585_0.15_0.85	9357	4021	106270.98	*9357*	*9357*	-	-	-	-	*9357*	*9357*	0	3.38
700_685_0.10_0.75	9881	5946	108642.02	*9881*	*9881*	-	-	-	-	*9881*	*9881*	0	2.26
700_685_0.15_0.85	9163	4488	128765.29	*9163*	*9163*	-	-	-	-	*9163*	*9163*	0	4.06
800_785_0.10_0.75	9937	2650	153807.41	*9937*	*9937*	-	-	-	-	*9937*	*9937*	0	8.61
800_785_0.15_0.85	9024	5379	126280.79	*9024*	8995.60	-	-	-	-	*9024*	**9024**	0	3.07
900_885_0.10_0.75	9725	4712	150334.35	*9725*	*9725*	-	-	-	-	*9725*	*9725*	0	1.25
900_885_0.15_0.85	8620	4714	163193.79	*8620*	8584.41	-	-	-	-	*8620*	**8619.68**	3.18	94.56
1000_985_0.10_0.75	9689	5283	171356.74	*9689*	9642.67	-	-	-	-	*9689*	**9689**	0	32.63
1000_985_0.15_0.85	8455	4501	194747.08	*8455*	8453.66	-	-	-	-	*8455*	**8455**	0	13.28
600_600_0.10_0.75	10524	6116	92516.76	*10524*	*10524*	-	-	-	-	*10524*	*10524*	0	0.90
600_600_0.15_0.85	9062	4256	101983.35	*9062*	**9062**	-	-	-	-	*9062*	9061.72	2.79	6.31
700_700_0.10_0.75	9786	5135	117038.18	*9786*	*9786*	-	-	-	-	*9786*	*9786*	0	1.99

Continued on next page

Table A.1 – continued from previous page

Instance	BKV	CPLEX		MSBTS [14]		PSO-B [16]		AES-BM [17]		ANRMA (this work)			
		LB	*UB*	f_{best}	f_{avg}	f_{best}	f_{avg}	f_{best}	f_{avg}	f_{best}	f_{avg}	*std*	t_{avg}
700_700_0.15_0.85	9229	5152	124816.91	*9229*	**9229**	-	-	-	-	*9229*	9228.92	0.80	10.49
800_800_0.10_0.75	9932	3990	133112.48	*9932*	*9932*	-	-	-	-	*9932*	*9932*	0	1.37
800_800_0.15_0.85	9101	5613	157381.61	*9101*	*9101*	-	-	-	-	*9101*	*9101*	0	3.73
900_900_0.10_0.75	9745	4350	145272.85	*9745*	*9745*	-	-	-	-	*9745*	*9745*	0	2.05
900_900_0.15_0.85	8990	5631	167528.40	*8990*	*8990*	-	-	-	-	*8990*	*8990*	0	5.01
1000_1000_0.10_0.75	9551	4557	188138.93	*9551*	*9551*	-	-	-	-	*9551*	*9551*	0	1.90
1000_1000_0.15_0.85	8538	0	202859.49	*8538*	8506.69	-	-	-	-	*8538*	**8538**	0	6.01
585_600_0.10_0.75	10393	6270	95625.13	*10393*	*10393*	-	-	-	-	*10393*	*10393*	0	1.05
585_600_0.15_0.85	9256	4530	100964.98	*9256*	*9256*	-	-	-	-	*9256*	*9256*	0	1.53
685_700_0.10_0.75	10121	5399	109236.96	*10121*	*10121*	-	-	-	-	*10121*	*10121*	0	1.05
685_700_0.15_0.85	9176	4514	127376.67	*9176*	*9176*	-	-	-	-	*9176*	*9176*	0	8.21
785_800_0.10_0.75	9384	5302	128560.22	*9384*	*9384*	-	-	-	-	*9384*	*9384*	0	2.11
785_800_0.15_0.85	8746	5213	150724.73	*8746*	8695.37	-	-	-	-	*8746*	**8746**	0	37.86
885_900_0.10_0.75	9318	4238	141827.11	*9318*	*9318*	-	-	-	-	*9318*	*9318*	0	5.16
885_900_0.15_0.85	8425	5335	167698.87	*8425*	8413.96	-	-	-	-	*8425*	**8425**	0	28.57
985_1000_0.10_0.75	9234	3989	187754.60	*9234*	9197.75	-	-	-	-	*9234*	**9234**	0	7.13
985_1000_0.15_0.85	8612	4809	193543.08	*8612*	8587.95	-	-	-	-	*8612*	**8612**	0	20.66
#Avg	10668.27	6969.92	95725.03	10668.27	10655.38	-	-	-	-	*10668.27*	**10667.24**	1.49	6.68

Table A.2 shows the original computational results of the proposed ANRMA algorithm and its comparisons with the references on the 54 medium benchmark instances of Set II. The meanings of each column and the indicators are the same as those in Table A.1.

Table A.2: Computational results of the ANRMA algorithm and reference algorithms on each of the 54 instances of the SUKP Set II.

Instance	BKV	CPLEX		MSBTS [14]		PSO-B [16]		AES-BM [17]		ANRMA (this work)			
		LB	*UB*	f_{best}	f_{avg}	f_{best}	f_{avg}	f_{best}	f_{avg}	f_{best}	f_{avg}	*std*	t_{avg}
600_500_0.10_0.70	13170	9177	180154.89	*13170*	*13170*	*13170*	13103	*13170*	13122.90	*13170*	*13170*	0	1.45
600_500_0.15_0.70	8007	5625	166951.20	*8007*	*8007*	7962	7705.20	7962	7779.20	*8007*	*8007*	0	0.88
600_500_0.10_0.75	15950	10835	195926.77	*15950*	15930.40	15760	15760	15762	15761.20	*15950*	**15950**	0	50.02
600_500_0.15_0.75	9611	5513	186133.92	*9611*	*9611*	9423	9411.80	9423	9423	*9611*	*9611*	0	1.82
600_500_0.10_0.80	20221	12005	207382.45	*20221*	20049.70	*20221*	*20221*	*20221*	*20221*	*20221*	*20221*	0	28.21
600_500_0.15_0.80	11771	7231	205343.29	*11771*	11768.40	11698	11698	11730	11719.40	*11771*	**11771**	0	14.55
800_500_0.10_0.70	14211	7462	244830.11	*14211*	14191.40	*14211*	14191.40	*14211*	14201.40	*14211*	**14211**	0	6.13
800_500_0.15_0.70	7801	5083	223084.53	*7801*	*7801*	7759	7739.20	*7801*	7761.50	*7801*	*7801*	0	2.56
800_500_0.10_0.75	17364	9179	252514.39	*17364*	*17364*	*17364*	17258.40	*17364*	17284.80	*17364*	*17364*	0	65.65
800_500_0.15_0.75	9415	5827	242355.67	*9415*	*9415*	9371	9270.60	9385	9299.90	*9415*	*9415*	0	139.30
800_500_0.10_0.80	22516	11341	286943.44	*22516*	21988.30	22049	21815.40	22058	21915.60	*22516*	**22516**	0	35.12
800_500_0.15_0.80	11818	6548	257127.05	*11818*	11793.60	11696	11655.90	*11818*	11671.10	*11818*	**11818**	0	4.80
1000_500_0.10_0.70	14296	6760	300407.53	14296	14292.50	14296	14229.80	14296	14238.80	**14336**	**14336**	0	62.47
1000_500_0.15_0.70	8327	3848	302080.63	*8327*	*8327*	8125	8102.20	*8327*	8134.40	*8327*	*8327*	0	1.49

Continued on next page

Table A.2 – continued from previous page

Instance	BKV	CPLEX		MSBTS [14]		PSO-B [16]		AES-BM [17]		ANRMA (this work)			
		LB	UB	f_{best}	f_{avg}	f_{best}	f_{avg}	f_{best}	f_{avg}	f_{best}	f_{avg}	std	t_{avg}
1000_500_0.10_0.75	17775	9386	325708.96	17586	17462.80	17752	17639.10	17760	17709.60	**17789**	**17789**	0	167.18
1000_500_0.15_0.75	9917	4973	319340.33	*9917*	*9917*	*9917*	9892.60	*9917*	*9917*	*9917*	*9917*	0	3.59
1000_500_0.10_0.80	23004	11016	342777.36	22752	22342.20	*23004*	22483	22854	22535.20	**23004**	**22988.80**	44.94	384.24
1000_500_0.15_0.80	12371	5968	331796.02	12371	12371	12330	12264.90	12361	12339.30	**12524**	**12524**	0	21.09
500_600_0.10_0.70	12207	8403	145447.11	*12207*	12198.20	12122	12053.20	12135	12124.60	*12207*	**12207**	0	7.42
500_600_0.15_0.70	7161	4302	141499.77	*7161*	*7161*	7043	7039	7097	7053.30	*7161*	*7161*	0	1.39
500_600_0.10_0.75	14703	10617	162337.11	*14703*	*14703*	14703	14652.30	*14703*	14654	*14703*	*14703*	0	4.23
500_600_0.15_0.75	8528	5994	150673.38	*8528*	*8528*	8347	8347	8414	8366.30	*8528*	*8528*	0	1.48
500_600_0.10_0.80	18270	11907	178212.45	*18270*	18149.20	18197	18111.70	*18270*	18157.80	*18270*	**18270**	0	3.54
500_600_0.15_0.80	10386	6689	162638.50	*10386*	10333.80	10234	10207.30	10250	10217.20	*10386*	**10386**	0	34.54
500_800_0.10_0.70	11357	7456	148036.64	*11357*	*11357*	*11357*	11256.80	*11357*	11285.70	*11357*	*11357*	0	0.88
500_800_0.15_0.70	6818	4412	140471.13	*6818*	*6818*	*6818*	*6818*	*6818*	*6818*	*6818*	*6818*	0	0.74
500_800_0.10_0.75	13593	8159	162685.42	*13593*	13571.80	13355	13122.20	13381	13330.10	*13593*	**13593**	0	1.99
500_800_0.15_0.75	7973	5095	153700.19	7973	*7973*	7938	7938	7940	7892.60	*7973*	**7973**	0	2.91
500_800_0.10_0.80	16551	11646	176303.13	*16551*	*16551*	16316	16197	16316	16248.90	*16551*	*16551*	0	2.98
500_800_0.15_0.80	9697	5298	170516.55	*9697*	*9697*	9684	9669.50	*9697*	9686.40	*9697*	*9697*	0	1.38
500_1000_0.10_0.70	10535	5888	154616.95	*10535*	*10535*	*10535*	10450.10	*10535*	10535	*10535*	*10535*	0	0.97
500_1000_0.15_0.70	6332	3549	145791.73	6190	6190	6332	6235	6190	6190	**6444**	**6444**	0	1.18
500_1000_0.10_0.75	12483	7338	167601.11	*12483*	*12483*	*12483*	12122.30	*12483*	12145.50	*12483*	*12483*	0	0.73
500_1000_0.15_0.75	7429	5337	154618.51	6980	6980	7428	7378.20	7429	7428.30	**7613**	**7613**	0	3.24
500_1000_0.10_0.80	15158	10214	179098.51	*15158*	*15158*	14892	14812.10	14965	14845.90	*15158*	*15158*	0	1.85
500_1000_0.15_0.80	8873	5961	169594.35	8873	8696	8773	8742.40	8773	8768.60	**9160**	**9160**	0	0.88
600_600_0.10_0.70	13058	8892	175358.89	*13058*	*13058*	12788	12649.50	13044	12892.50	*13058*	*13058*	0	0.99
600_600_0.15_0.70	7454	4670	178962.34	*7454*	*7454*	*7454*	7242.60	*7454*	7361.40	*7454*	*7454*	0	0.76
600_600_0.10_0.75	15505	10602	199867.70	*15505*	*15505*	15383	15383	*15505*	15468.40	*15505*	*15505*	0	2.14
600_600_0.15_0.75	8914	4888	193940.85	*8914*	*8914*	8765	8739.80	8765	8765	*8914*	*8914*	0	1.39
600_600_0.10_0.80	18994	11533	210008.85	*18994*	18988	*18994*	18940.20	*18994*	18956.70	*18994*	**18994**	0	3.92
600_600_0.15_0.80	10970	6503	207874.24	*10970*	*10970*	10848	10740.50	10848	10820.80	*10970*	*10970*	0	1.83
800_800_0.10_0.70	11867	3837	252869.56	*11867*	*11867*	11853	11544	11853	11777.60	*11867*	*11867*	0	1.68
800_800_0.15_0.70	6901	3987	253577.27	6882	6882	6882	6873.70	6901	6884.30	**7083**	**7083**	0	12.09
800_800_0.10_0.75	14351	7969	262408.50	*14351*	*14351*	14125	14045.10	14186	14162.50	*14351*	*14351*	0	1.90
800_800_0.15_0.75	8196	5340	259819.34	8196	8196	8196	8137.50	8196	8196	**8384**	**8384**	0	9.65
800_800_0.10_0.80	17667	9077	287888.51	*17667*	*17667*	*17667*	17544.70	*17667*	17620.60	*17667*	*17667*	0	25.82
800_800_0.15_0.80	9986	5891	266093.24	9986	9986	9798	9642.80	9920	9809.10	**10107**	**10105.90**	3	248.23
1000_1000_0.10_0.70	11306	6815	346829.11	11306	11168.80	11203	11081.40	11217	11193.70	**11308**	**11308**	0	74.83
1000_1000_0.15_0.70	6766	4527	339808.31	*6766*	6687.70	6763	6694.50	6763	6763	*6766*	**6766**	0	1.86
1000_1000_0.10_0.75	13291	7468	373137.20	13291	13259	12983	12960.40	13122	12992.60	**13511**	**13511**	0	95.62
1000_1000_0.15_0.75	8027	4743	339667.52	*8027*	7972.20	7683	7630.60	7683	7683	*8027*	**8027**	0	2.44
1000_1000_0.10_0.80	16428	10156	366036.22	*16428*	16308.20	15913	15835.30	16044	15947.90	*16428*	**16428**	0	30.68
1000_1000_0.15_0.80	9465	0	380351.29	9465	9465	9428	9230.70	9443	9441.50	**9644**	**9644**	0	16.35
Avg		12124.91	7091.48	22831	8.52	12105.44	12066.37	12025.76	11935.37	12051.44	11991.11	**12156.06**	**12155.75**	0.89	29.54

B Summary of comparative results between ANRMA and reference algorithms

In order to highlight the superiority of ANRMA, Table B.1 summarizes the comparative results between ANRMA and each reference algorithm along with the best-known values (BKV). The first column gives the pairs of two algorithms (or values) compared, while the respective instance sets are listed in the second column. The third column indicates the main indicators f_{best} and f_{avg}. The fourth to the sixth columns summarize the counts of each kind of comparison results, where *Wins*, *Ties* and *Losses* respectively represent obtaining a better, equal and worse result of f_{best} or f_{avg}. The *p-value* shows the results of Wilcoxon signed-rank test presented in the last column, where 'NA' indicates two corresponding groups have exactly the same results.

Table B.1
Summary of comparisons between the proposed ANRMA algorithm and state-of-the-art SUKP algorithms.

Algorithm pair	Instance set	Indicator	*Wins*	*Ties*	*Losses*	*p-value*
ANRMA vs. MSBTS [14]	Set I (60/60)	f_{best}	0	60	0	NA
		f_{avg}	22	35	3	7.22e-05
	Set II (54)	f_{best}	13	41	0	1.47e-03
		f_{avg}	27	27	0	5.60e-06
ANRMA vs. PSO-B [16]	Set I (30/60)	f_{best}	0	30	0	NA
		f_{avg}	11	18	1	3.70e-03
	Set II (54)	f_{best}	40	14	0	3.56e-08
		f_{avg}	52	2	0	3.50e-10
ANRMA vs. AES-BM [17]	Set I (30/60)	f_{best}	0	30	0	NA
		f_{avg}	3	25	2	3.45e-01
	Set II (54)	f_{best}	35	19	0	2.47e-07
		f_{avg}	50	4	0	7.56e-10
ANRMA vs. I2PLS [12]	Set III (18)	f_{best}	18	0	0	7.63e-06
		f_{avg}	18	0	0	7.63e-06
ANRMA vs. ATS-DLA [13]	Set III (18)	f_{best}	3	15	0	1.09e-01
		f_{avg}	7	6	5	3.47e-01
ANRMA vs. MSBTS [14]	Set III (18)	f_{best}	12	6	0	2.20e-03
		f_{avg}	16	2	0	4.38e-04
ANRMA vs. BKV	Set I (60/60)	f_{best}	0	60	0	NA
		f_{avg}	12	44	4	3.78e-03
	Set II (54)	f_{best}	12	42	0	2.22e-03
		f_{avg}	26	28	0	8.30e-06
	Set III (18)	f_{best}	2	16	0	1.80e-01
		f_{avg}	7	6	5	7.54e-01

From Table B.1, we can observe that ANRMA attains either better or equal f_{best} results when compared to the references. Even when compared with ATS-DLA, an algorithm specifically designed for large instances, ANRMA also consistently achieves better or equal f_{best} results without exception. Besides, ANRMA achieves a majority of better f_{avg} results when compared

to the references. The small *p-value* results (< 0.05) of the Wilcoxon signed-rank test on most comparisons also confirm the significant differences between the results. When comparing to the BKV reported in previous studies, ANRMA still shows its competitiveness in obtaining 14 new best f_{best} results and improving or achieving most of the f_{avg} results.

C Computational results on the BMCP instances of Sets I and II

Table C.1 shows the original computational results of the proposed ANRMA algorithm and the references on the 30 benchmark instances of the BMCP Set I. The meanings of each column and the indicators are the same as those in Table A.1.

Table C.1: Computational results of the ANRMA algorithm and reference algorithms on each of the 30 instances of the BMCP Set I.

Instance	BKV	PLTS [7]		VDLS [35]		IHS [8]		ANRMA (this work)		
		f_{best}	f_{avg}	f_{best}	f_{avg}	f_{best}	f_{avg}	f_{best}	f_{avg}	$stdt_{avg}$
585_600_0.05_2000	71102	*71102*	71065	*71102*	71034	*71102*	71097	*71102*	**71102**	0 0.96
585_600_0.075_1500	71025	70677	70677	*71025*	70493	*71025*	*71025*	*71025*	*71025*	0 0.88
600_585_0.05_2000	67636	*67636*	67461	*67636*	67622	*67636*	*67636*	*67636*	*67636*	0 0.73
600_585_0.075_1500	70588	*70588*	70407	*70588*	70277	*70588*	*70588*	*70588*	*70588*	0 1.31
600_600_0.05_2000	68738	*68738*	68472	*68738*	68581	*68738*	*68738*	*68738*	*68738*	0 0.64
600_600_0.075_1500	71904	71746	71746	*71904*	71781	*71904*	*71904*	*71904*	*71904*	0 1.03
685_700_0.05_2000	81227	*81227*	80586	*81227*	80483	*81227*	*81227*	*81227*	*81227*	0 1.06
685_700_0.075_1500	83286	82955	82951	*83286*	82887	*83286*	83176	*83286*	**83286**	0 1.10
700_685_0.05_2000	78054	*78054*	78037	*78054*	77743	*78054*	*78054*	*78054*	*78054*	0 1.05
700_685_0.075_1500	78869	*78869*	*78869*	*78869*	78643	*78869*	*78869*	*78869*	*78869*	0 0.96
700_700_0.05_2000	78458	78028	77859	*78458*	78180	*78458*	*78458*	*78458*	*78458*	0 1.40
700_700_0.075_1500	84576	*84576*	84376	*84576*	*84576*	*84576*	*84576*	*84576*	*84576*	0 0.80
785_800_0.05_2000	92740	92608	92588	*92740*	92414	*92740*	*92740*	*92740*	*92740*	0 3.39
785_800_0.075_1500	95221	94245	94245	*95221*	*95221*	*95221*	*95221*	*95221*	*95221*	0 0.74
800_785_0.05_2000	89138	*89138*	88581	*89138*	89055	*89138*	*89138*	*89138*	*89138*	0 2.39
800_785_0.075_1500	91856	91021	91010	*91856*	91723	*91856*	*91856*	*91856*	*91856*	0 1.31
800_800_0.05_2000	91795	*91795*	91576	*91795*	91573	*91795*	*91795*	*91795*	*91795*	0 0.97
800_800_0.075_1500	95995	95533	95510	*95995*	95569	*95995*	*95995*	*95995*	*95995*	0 1.45
885_900_0.05_2000	102277	102162	101332	*102277*	102059	*102277*	*102277*	*102277*	*102277*	0 2.06
885_900_0.075_1500	106940	106577	105942	*106940*	106447	*106940*	*106940*	*106940*	*106940*	0 2.55
900_885_0.05_2000	99590	98840	98718	*99590*	99365	*99590*	*99590*	*99590*	*99590*	0 0.75
900_885_0.075_1500	105141	*105141*	104398	*105141*	105025	*105141*	*105141*	*105141*	*105141*	0 1.11
900_900_0.05_2000	102055	101265	101231	*102055*	101403	*102055*	101728	*102055*	**102055**	0 0.98
900_900_0.075_1500	105081	104521	104521	*105081*	104919	*105081*	*105081*	*105081*	*105081*	0 1.22
985_1000_0.05_2000	110669	109567	109409	*110669*	109793	*110669*	*110669*	*110669*	*110669*	0 1.41
985_1000_0.075_1500	115505	114969	113838	*115505*	114765	*115505*	*115505*	*115505*	*115505*	0 0.82
1000_985_0.05_2000	112057	111859	111229	*112057*	111583	*112057*	*112057*	*112057*	*112057*	0 2.30
1000_985_0.075_1500	113615	112250	112126	*113615*	113096	*113615*	*113615*	*113615*	*113615*	0 1.05

Continued on next page

Table C.1 – continued from previous page

Instance	BKV	PLTS [7]		VDLS [35]		IHS [8]		ANRMA (this work)			
		f_{best}	f_{avg}	f_{best}	f_{avg}	f_{best}	f_{avg}	f_{best}	f_{avg}	$stdt_{avg}$	
1000_1000_0.05_2000	113331	112802	111897	*113331*	113125	*113331*	113316	*113331*	**113331**	0	1.05
1000_1000_0.075_1500	120246	*120246*	118468	*120246*	119995	*120246*	*120246*	*120246*	*120246*	0	1.41
Avg	92291	91958	91637	*92291*	91981	*92291*	92275	*92291*	**92291**	0	1.30

Table C.1 shows the original computational results of the proposed ANRMA algorithm and the references on the 60 benchmark instances of the BMCP Set II. The meanings of each column and the indicators are the same as those in Table A.1.

Table C.2: Computational results of the ANRMA algorithm and reference algorithms on each of the 60 instances of the BMCP Set II.

Instance	BKV	PLTS[7]		VDLS[35]		IHS [8]		ANRMA (this work)			
		f_{best}	f_{avg}	f_{best}	f_{avg}	f_{best}	f_{avg}	f_{best}	f_{avg}	std	t_{avg}
1100_1000_0.3_3000	143475	142835	142429	*143475*	143225	*143475*	*143475*	*143475*	*143475*	0	1.53
1100_1000_0.5_2000	126917	125974	125790	126917	126022	*127157*	*127157*	*127157*	*127157*	0	2.73
1100_1100_0.3_3000	157530	154936	154662	*157530*	156463	*157530*	*157530*	*157530*	*157530*	0	3.25
1100_1100_0.5_2000	139196	139196	138169	139141	138719	*139235*	*139235*	*139235*	*139235*	0	6.21
1100_1200_0.3_3000	172660	169113	168764	*172660*	172075	*172660*	*172660*	*172660*	*172660*	0	1.16
1100_1200_0.5_2000	150493	150301	149366	*150493*	149944	*150493*	150464	*150493*	**150493**	0	3.35
1200_1100_0.3_3000	161848	*161848*	159620	*161848*	*161848*	*161848*	*161848*	*161848*	*161848*	0	1.84
1200_1100_0.5_2000	137700	135858	135509	*137700*	136831	*137700*	137635	*137700*	**137700**	0	24.59
1200_1200_0.3_3000	172595	169219	168822	*172595*	172152	*172595*	**172595**	*172595*	172581	77.55	118.96
1200_1200_0.5_2000	151440	149654	148679	*151440*	149610	*151440*	**151440**	*151440*	151420	105.91	83.63
1200_1300_0.3_3000	186083	183004	182220	*186083*	183599	*186083*	*186083*	*186083*	*186083*	0	4.82
1200_1300_0.5_2000	162154	160369	159735	*162154*	161147	*162154*	*162154*	*162154*	*162154*	0	31.41
1300_1200_0.3_3000	170786	168586	168474	*170786*	169725	*170786*	170782	*170786*	**170786**	0	2.46
1300_1200_0.5_2000	151402	151067	150035	151402	151049	*151512*	*151512*	*151512*	*151512*	0	7.17
1300_1300_0.3_3000	184699	180608	179978	*184699*	183239	*184699*	*184699*	*184699*	*184699*	0	3.78
1300_1300_0.5_2000	162146	160485	159511	162146	161537	*162514*	162211	*162514*	**162514**	0	142.39
1300_1400_0.3_3000	199134	197709	195682	*199134*	198889	*199134*	*199134*	*199134*	*199134*	0	7.58
1300_1400_0.5_2000	173954	173349	171665	*173954*	173799	*173954*	*173954*	*173954*	*173954*	0	119.44
1400_1300_0.3_3000	183987	182881	180144	183987	183796	*184691*	184673	*184691*	**184691**	0	13.68
1400_1300_0.5_2000	162544	161995	160647	*162544*	162313	*162544*	162521	*162544*	**162544**	0	56.60
1400_1400_0.3_3000	200136	199879	197229	*200136*	199617	*200136*	*200136*	*200136*	*200136*	0	4.53
1400_1400_0.5_2000	175583	174683	173393	*175583*	173985	*175583*	175477	*175583*	**175583**	0	32.53
1400_1500_0.3_3000	212492	210077	208316	*212492*	211264	*212492*	*212492*	*212492*	*212492*	0	24.85
1400_1500_0.5_2000	186818	184259	183290	*186818*	186250	*186818*	186811	*186818*	*186818*	0	33.74
1500_1400_0.3_3000	197219	194156	193555	197219	196703	*197899*	*197899*	*197899*	*197899*	0	5.01
1500_1400_0.5_2000	174284	173475	171427	174284	173576	**174522**	**174522**	174370	174367	15.44	396.51
1500_1500_0.3_3000	211108	209929	207795	*211108*	210394	*211108*	*211108*	*211108*	*211108*	0	21.02
1500_1500_0.5_2000	187052	184082	182676	*187052*	185222	*187052*	187000	*187052*	**187052**	0	13.47
1500_1600_0.3_3000	226408	*226408*	221641	*226408*	224927	*226408*	*226408*	*226408*	*226408*	0	4.27

Continued on next page

Table C.2 – continued from previous page

Instance	BKV	PLTS [7]		VDLS [35]		IHS [8]		ANRMA (this work)			
		f_{best}	f_{avg}	f_{best}	f_{avg}	f_{best}	f_{avg}	f_{best}	f_{avg}	std	t_{avg}
1500_1600_0.5_2000	199678	198214	196659	*199678*	198463	*199678*	199449	*199678*	**199669**	49.18	84.93
4200_4000_0.3_10000	1835947	1778013	1747412	*1835947*	1833108	*1835947*	*1835947*	*1835947*	*1835947*	0	41.88
4200_4000_0.5_7000	1738553	1685141	1666673	1738553	1729188	1737928	1737928	**1738553**	**1738216**	550.33	711.22
4200_4200_0.3_10000	1925706	1874190	1839780	1925706	1918624	*1925859*	*1925859*	*1925859*	*1925859*	0	65.13
4200_4200_0.5_7000	1811649	1764277	1742122	1811649	1801880	1810670	1810670	**1816714**	**1812620**	1856.14	1074.69
4200_4400_0.3_10000	2023093	1954069	1932251	2023093	2020210	2022834	2022834	**2023093**	**2023067**	77.70	555.21
4200_4400_0.5_7000	1903515	1853677	1820444	1903515	1889810	1892941	1892941	**1903515**	**1903493**	108.68	736.02
4400_4200_0.3_10000	1920418	1833465	1816879	*1920418*	1911396	*1920418*	*1920418*	*1920418*	*1920418*	0	678.22
4400_4200_0.5_7000	1810031	1746064	1730784	1810031	1805201	*1811179*	**1811179**	*1811179*	1811075	209.80	804.89
4400_4400_0.3_10000	2027801	1945218	1922134	2027801	2024036	*2028288*	*2028288*	*2028288*	*2028288*	0	252.21
4400_4400_0.5_7000	1901941	1844031	1821468	1901941	1888301	*1905810*	**1905810**	*1905810*	1900511	1976.05	1510.05
4400_4600_0.3_10000	2100925	2003959	1993946	2100925	2093610	*2100925*	*2100925*	*2100925*	*2100925*	0	331.57
4400_4600_0.5_7000	1989582	1937066	1906956	1989582	1979417	*1991870*	**1991870**	*1991870*	1989460	1398.62	702.59
4600_4400_0.3_10000	2008258	1930498	1913501	2008258	2001369	*2011730*	**2011730**	*2011730*	2011639	489.33	550.84
4600_4400_0.5_7000	1898602	1845046	1817697	1898602	1889659	*1901834*	**1901834**	*1901834*	1900506	1357.53	645.10
4600_4600_0.3_10000	2106938	2027132	2005097	2106938	2100125	2106938	2106938	**2106970**	**2106970**	0	176.61
4600_4600_0.5_7000	1980462	1921261	1900376	1980462	1973677	1981464	**1981464**	**1984295**	1980501	1569.26	1627.10
4600_4800_0.3_10000	2201977	2120120	2091723	2201977	2197572	*2203833*	*2203833*	*2203833*	*2203833*	0	77.55
4600_4800_0.5_7000	2070913	1989454	1967943	2070913	2057914	2066426	2066426	**2071179**	**2069204**	1382.44	720.63
4800_4600_0.3_10000	2096747	2019033	1991674	2096747	2087482	*2101148*	**2101148**	*2101148*	2101130	95.32	454.26
4800_4600_0.5_7000	1991409	1922466	1901158	1991409	1973722	1984983	1981122	**1992370**	**1989253**	4457.55	605.68
4800_4800_0.3_10000	2183525	2100773	2058922	2183525	2175465	2185336	2185336	**2185367**	**2185367**	0	234.92
4800_4800_0.5_7000	2066487	2009129	1973994	2066487	2059459	2072811	**2072811**	**2075133**	2070586	2081.69	119.06
4800_5000_0.3_10000	2280317	2184445	2157793	2280317	2273869	*2280317*	*2280317*	*2280317*	*2280317*	0	187.38
4800_5000_0.5_7000	2146413	2087596	2050741	2146413	2136313	2149938	**2149156**	**2151715**	2146206	2111.22	1622.73
5000_4800_0.3_10000	2183483	2069979	2051607	2183483	2175171	*2184508*	**2184508**	*2184508*	2183556	656.82	1201.67
5000_4800_0.5_7000	2066000	2013921	1977311	2066000	2053360	2060541	2060541	**2070143**	**2067754**	923.40	1300.24
5000_5000_0.3_10000	2261620	2144223	2126463	2261620	2251527	2260089	2260089	**2261620**	**2261444**	661.64	490.22
5000_5000_0.5_7000	2151921	2067188	2046887	2151921	2141829	**2151921**	**2151328**	2151605	2148973	1648.54	654.05
5000_5200_0.3_10000	2354408	2231359	2220887	2354408	2345759	*2354408*	**2354408**	*2354408*	2354299	409.78	615.70
5000_5200_0.5_7000	2222169	2129556	2115338	2222169	2211651	2222527	2222527	**2225839**	**2223453**	2040.95	681.37
Avg	1108039	1070108	1057431	1108038	1103285	1108155	1108054	**1108999**	**1108309**	438.51	344.77