2023 IEEE Symposium Series on Computational Intelligence (SSCI)
Mexico City, Mexico. December 5-8, 2023

Fixed set search applied to the maximum set k-covering problem

Raka Jovanovic
Qatar Environment and Energy Research Institute
Hamad bin Khalifa University
Doha, Qatar
rjovanovic@hbku.edu.qa

***Abstract*—The MKCP (Maximum Set k-Covering Problem) is a widely recognized combinatorial problem that falls under the category of NP-hard problems. It has diverse applications and involves the goal of covering a maximum number of elements using a limited number of candidate sets. In this paper, the novel fixed set search (FSS), a population based metaheuristic, is applied on the problem of interest. The FSS adds a learning mechanism to the greedy randomized adaptive search procedure (GRASP) based on elements frequently occurring in high quality solutions. The main advantage of the proposed approach is the simplicity of implementation compared to the current state-of-the-art methods. The conducted computational experiments show that the FSS even when using a simple local search manages to be highly competitive to state-of-the-art methods. In addition, the FSS manages to find two new lower bounds for the standardly used benchmark test instances. Finally, the performed computational experiments show that the learning mechanism of the FSS significantly improves the performance of the underlying GRASP algorithm.**

***Index Terms*—set covering problem, fixed set search, GRASP**

I. Introduction

The minimum set cover problem (MSCP) is a well-established challenge in the realm of combinatorial optimization. Its primary objective is to identify the set cover that employs the fewest sets [1]. In the maximum set k-covering problem (MKCP), the aim is to discover a subset of sets with a specified cardinality that maximizes the number of elements covered by this particular subset. The MKCP is widely acknowledged as a challenging problem classified under the NP-hard category [1]. Its applications extend across numerous industrial engineering domains, showcasing its versatility and practicality. Examples of such applications include the maximum covering location problem [2], wireless sensor networks [3], cloud computing [4], train diver scheduling [5], among many others. These diverse real-world implementations underscore the significance and wide-ranging impact of the MKCP in industrial engineering scenarios.

The computational complexity of MKCP makes it infeasible to rely on exact algorithms for handling large-scale instances. Consequently, researchers have invested significant efforts in developing heuristics to obtain high-quality solutions. An effective greedy method based on one-pass streaming algorithms has been proposed in [6]. For the MKCP, solution methods using local searches have proven to be highly effective. One such example is the restart local search algorithm (RNKC) [7]. The best performing methods for the MKCP combine different metaheuristics with local searches. Some examples are the use of adaptive binary particle optimization [8] and the binary artificial bee colony algorithm [9]. To the best of the authors knowledge, currently the best performing method is the max–min ant system with memory ants [10]. It should be noted that all these population based methods [8]–[10] use more advanced local searches than the basic one based on the swap operation (removing one set from the set cover and adding a new one) to maximize the performance.

The Fixed Set Search (FSS) is an innovative metaheuristic that has effectively tackled various optimization problems. It has been successfully employed to solve the traveling salesman problem [11], power dominating set problem [12], machine scheduling [13], clique partitioning problem [14], minimum weighted vertex cover problem [15], covering location with interconnected facilities problem [16] and others. Additionally, the FSS has demonstrated its efficacy in addressing bi-objective optimization problems [17]. The FSS is a population-based metaheuristic that incorporates a local search, making it highly suitable for the MKCP, as methods utilizing local searches have proven most successful. In practice, the FSS integrates a learning mechanism into the Greedy Randomized Adaptive Search Procedure (GRASP) metaheuristic [18]. FSS is motivated by the observation that high-quality solutions for a given problem instance often share common elements. The approach focuses on generating solutions that incorporate these elements, known as the fixed set. The computational effort is then directed towards completing the partial solutions by "filling in the gaps."

This paper focuses on the application of the FSS on the MKCP. One of the main advantages of the FSS is that its learning mechanism can drastically improve the performance of the GRASP especially for less effective local searches [11], [15]. One of the disadvantages of the best performing methods [8]–[10] is that they become significantly more complex for implementation which the proposed FSS approach tries to avoid. To be exact, the proposed FSS uses the simple swap based neighborhood in the local search and relies on the FSS learning mechanism for enhancing performance. The effectiveness of the proposed approach is tested on the standard benchmark test set and proves to significantly outperform other simple to implement methods and is highly competitive to the more complex state-of-the-art ones.

The paper is organized as follows. The Section II provides the problem formulation for the MKCP. The following three sections are dedicated to the greedy algorithm, local search and GRASP algorithm for the problem of interest. The Section VI focuses on the FSS method. The next section shows the results of the conducted computational experiments and provides their analysis. The paper is finalized with some concluding remarks.

II. Problem formulation

The formal definition of MKCP [1] is as follows: Consider $R = \{1, \ldots, r\}$ a set of rows and $C = \{1, \ldots, c\}$ a set of columns. There is a binary matrix, denoted as $A = (a_{ij})$, with r rows and c columns. A column j covers a row i if the entry a_{ij} equals 1. The objective of MKCP is to select a subset $S \subset C$ of size k ($|S| = k$), maximizing the number of covered rows.

The integer programming model (IP) can be specified as follows [9]. Let us define binary decision variables x_j for $j \in C$. The value of $x_j = 1$ is used for all for a column $j \in S$ and zero otherwise. In addition, lets us define auxiliary binary variables y_i, for $i \in R$, which are equal to 1 if set S covers row i and 0 otherwise. Using these variables the IP can be specified using the following equations.

$$\text{Maximize} \sum_{i \in R} y_j \tag{1}$$

$$\text{Subjected to}$$

$$\sum_{j \in C} x_j = k \tag{2}$$

$$\sum_{j \in C} a_{ij} x_j \geq y_j \qquad j \in R \tag{3}$$

$$x_j \in \{0, 1\} \qquad j \in C \tag{4}$$

$$y_i \in \{0, 1\} \qquad i \in R \tag{5}$$

The Eq. (1) states that objective is to maximize the total number of covered rows. The constraint given in Eq. (2) guaranties that the there are k selected columns in the solution S. The constraints given in Eq.(3), guarantee that a row can only be covered if at least one column that covers it is selected. Finally, Eqs. (4) and (5) provide the bounds on the variables related to selected columns and covered rows, respectively.

III. Greedy algorithm

The idea of the greedy algorithm is to start with an empty partial solution $S = \emptyset$ and iteratively expand it. At each iteration, the partial solution S is expanded by a column $o \in S^c$, where $S^c = C \setminus S$, which is not already a part of S. Since our goal is to have the maximal covering, the selected element for expansion o should cover the largest number of rows that are not already covered by some column in S.

In relation, let us define the function $Cover(S)$, for a set of columns S, as the number of rows covered by some column in S. Now, the heuristic function h, for a partial solution S and a column o, can be specified as follows.

$$h(S, o) = Cover(S \cup \{o\}) - Cover(S) \tag{6}$$

In Eq. (6) presents the heuristic function $h(S, o)$ as the number of newly covered rows in the case set S is expanded with the column o. Using function h, the greedy algorithm can be fully specified. Since, our goal is to use the greedy algorithm as a part of the GRASP metaheuristic, it is necessary to include randomization. In the proposed algorithm, we use the standard approach of a restricted candidate list (RCL) to achieve this as follows. Let us define L as the set on α elements $o \in S^c$ that have the largest value of $h(S, o)$. Now, we can expand the partial solution with a random element of set L instead of the one having the largest value of the heuristic function h. The details of the proposed randomized greedy algorithm can be seen in Alg. 1.

Algorithm 1 Randomized greedy algorithm for the MKCP

```
Input: Problem instance A, Size of RCL $\alpha$
Output: Generated covering $S$
function RandomizedGreedy(A, $\alpha$)
    $S = \emptyset$; $S^c = Columns(A)$
    while $|S| < k$ do
        $o = RCL(S, \alpha)$
        $S = S \cup \{o\}$
        $S^c = S^c \setminus \{o\}$
    end while
    return $S$
end function
```

IV. Local search

The local search is based on the concept of swapping elements of a solution S with elements of $S^c = C \setminus S$. The idea is to swap an element of a solution $i \in S$ with one outside of it ($o \in S^c$) if the resulting solutions covers more rows. This procedure is repeated until not further improvement can be archived.

To formally define this approach, let us define the function $Swap(S, i, o)$ that is equal to the change in the number of covered rows when column i is removed from S and column $o \in S^c$ is added, as follows

$$Swap(S, i, o) = Cover((S \setminus \{i\}) \cup \{o\}) - Cover(S) \tag{7}$$

In relation, let us define the $Imp(S)$ as the set of all improving swap operations for a solution S.

$$Imp(S) = \{(i, o) \mid i \in S \wedge o \in S^c \wedge Swap(S, i, o) > 0\} \tag{8}$$

In Eq. (8), $Imp(S)$ represented the set of all swap operations, specified using the pairs (i, o) where column i is removed from S and column o is added, that increase the number of covered rows. Using the set of improving swap operations $Imp(S)$ the local search can be specified, for which details can be seen in Alg. 2.

In Alg. 2, a solutions S is iteratively improved using the following procedure. While the set of improving solutions $Imp(S)$ is not empty, a random swap operation using columns $i \in S$ and $o \in S^c$ is selected and applied to S.

Algorithm 2 Local search for the MKCP

```
Input: Problem instance $A$, Initial solution $S$
Output: Improved solution $S$
function LocalSearch(A, S)
    while $Imp(S) \neq \emptyset$ do
        Select random $(i, o) \in Imp(S)$
        $S = (S \setminus \{i\}) \cup \{o\}$
    end while
    return $S$
end function
```

V. GRASP

To improve the efficiency of the proposed method the greedy algorithm and local search are incorporated into the GRASP metaheuristic. The GRASP iteratively generates solutions using a randomized greedy algorithm and applies a local search on each of them. The details of the GRASP used as a part of the FSS is depicted in Algorithm 3. Firstly, the set of generated solutions $\mathcal{S}$ is set to an empty set. Within the main loop of Alg 3, a fresh solution S for the MKCP is generated using the *RandomizedGreedy* function. The local search is then performed on the solution S, and it is added to the set of generated solutions $\mathcal{S}$ and is evaluated if it qualifies as the new best solution. This process continues iteratively until a certain stopping criterion is met, typically defined by a time limit or reaching the maximum allowed number of generated solutions.

Algorithm 3 Pseudocode for the GRASP

```
Input: Problem instance $A$, Size of RCL $\alpha$, Number of iterations $MaxIter$
Output: Set Of generated solutions $\mathcal{S}$
function GRASP(A, $\alpha$, MaxIter)
    i=0; $\mathcal{S} = \emptyset$
    while $i < MaxIterations$ do
        $S = RandomizedGreedy(A, \alpha)$
        $S = LocalSearch(A, S)$
        $\mathcal{S} = \mathcal{S} \cup S$
        i = i+1
    end while
    return $\mathcal{S}$
end function
```

VI. Fixed Set Search

In this section, the application of the FSS metaheuristic to the MKCP is presented. As previously stated, the FSS combines GRASP with a learning mechanism to identify common elements in high-quality solutions and generates new solutions containing them. The algorithm includes a method for generating fixed sets, using a greedy algorithm with local search, and implementing a learning mechanism. The process involves generating an initial population, creating fixed sets, and iteratively generating new solutions with the selected elements. In the following subsections each of these components is described.

A. Fixed Set Generation

The first building block involves representing a solution as a subset derived from a ground set of elements. As previously mentioned, a MKCP solution S can be seen as a subset composed of elements (columns) $S \subset C$. The second building block is a method for generating multiple fixed sets F while allowing control over their size or cardinality $|F|$. Moreover, when incorporating a generated fixed set into the randomized greedy algorithm, it must be capable of producing feasible solutions of equal or superior quality compared to those generated by the underlying greedy algorithm. To establish the foundation, let us introduce some definitions.

We denote $\mathcal{S}_n = \{S_1, .., S_n\}$ as the set of the n best solutions generated in the preceding algorithm steps. A base solution $B \in \mathcal{S}_n$ is randomly selected from the top n solutions. If the fixed set fulfills the condition $F \subset B$, it can be utilized to generate a feasible solution with a quality equal to or better than that of B. Furthermore, F can incorporate an arbitrary number of elements from B. The underlying concept is to include elements in F that occur frequently within a group of high-quality solutions. We define $\mathcal{S}_{kn}$ as the set consisting of k randomly selected solutions from the n best ones, denoted as $\mathcal{S}_n$. By leveraging these defined components, we can effectively generate fixed sets tailored for the MKCP.

Let us introduce the function $C(e, S)$, which operates on a solution S and an element (column) e. It evaluates to 1 if e is present in S, and 0 otherwise. Leveraging this function, we can calculate the occurrence count of an element e within $\mathcal{S}_{kn}$ using the expression:

$$O(e, \mathcal{S}_{kn}) = \sum_{S \in \mathcal{S}_{kn}} C(e, S) \tag{9}$$

Subsequently, we define F as a subset of B consisting of the elements e with the highest values of $O(e, \mathcal{S}_{kn})$. Furthermore, we denote the process of generating the fixed set, given a base solution B and a set of solutions $\mathcal{S}_{kn}$ with a specified size $Size$, as the function $F = Fix(B, \mathcal{S}_{kn}, Size)$. Note that in the proposed approach for generating fixed sets, ties between elements are broken randomly.

B. Randomized greedy algorithm with pre-selected elements

An extension of the randomized greedy algorithm discussed in Section III, can be easily adapted to incorporate a fixed set F. In this adaptation, the initial partial solution S is set to F instead of an empty set. To distinguish this modified version of the algorithm, we denote it as *RandomizedGreedyWithFix*(F, A, α), where F represents the pre-selected set of elements.

C. Learning mechanism

This section describes the FSS learning mechanism, specifically focusing on utilizing fixed sets to leverage experience from previous solutions. Initially, N solutions are generated using the GRASP as the initial population $\mathcal{S}$ for exploring the solution space. Next, the FSS iteratively generates solutions as follows: a fixed set F of size $Size$ is generated using previous solutions, and it is used with the randomized greedy algorithm to create a new solution S. The local search is applied to improve S, and the newly obtained locally optimal solution is added to the set of solutions $\mathcal{S}$. This process continues until a stopping criterion is met.

The fixed set plays a vital role in the algorithm. It is generated using a base solution and a set of test solutions. The fixed set's size is incrementally increased when stagnation occurs after, many iterations without finding a new solutions among the best n solutions $\mathcal{S}_n$. However, to avoid repetitive solutions, an upper bound is set on the fixed set size. If this bound is reached during stagnation, the size is reset to the minimum allowed value. This cycle continues until the stopping criterion is reached.

To determine the sizes of the fixed sets in the FSS learning mechanism, an array of permissible sizes is established. These sizes are defined relative to the base solutions' size. In the proposed implementation, the array is defined as:

$$Portion[i] = \left(1 - \beta^i\right) \quad (10)$$

In Eq. 10 the value $\beta \in (0, 1)$ is used. The size of the fixed set is proportional to the base solution B cardinality. At the i-th level, the size is set to $\lfloor |B| \cdot Portion[i] \rfloor$ (rounded down). Here, the base solution's size corresponds to the number of columns in the solution. The maximum permissible size of the fixed set should satisfy the condition $|B|Portion[i] \leq f$, where f is a predetermined number.

The algorithmic representation of the FSS is illustrated in Algorithm 4. In this algorithm, the first step involves the initialization of fixed set sizes using Eq. (10). The size of the current fixed set, denoted as $Size$, is initially set to the smallest value. The subsequent part of the initialization phase generates the initial population of solutions by executing N iterations of the basic GRASP algorithm. Each iteration of the main loop consists of the following steps: first, a random set of solutions $\mathcal{S}_{kn}$ is generated by selecting k elements from $\mathcal{S}_n$, and a random base solution B is chosen from the set $\mathcal{S}_n$. Next, the function $Fix(B, \mathcal{S}_{kn}, Size|B|)$ is utilized to generate a fixed set F. Subsequently, a new solution $S = RandomizedGreedyWithFix(F, A, \alpha)$ is created using the randomized greedy algorithm with pre-selected elements, followed by the application of a local search. The algorithm then checks whether S improves the best solution and adds it to the set of generated solutions $\mathcal{S}$. If stagnation occurs, the value of $Size$ is updated to the next value in the array $Sizes$. Here, stagnation is considered if in the last M iterations no new solution has been added to the set of S_n best solutions. It is noteworthy that the next size corresponds to the next larger element in the $Sizes$ array. In the case where $Size$ already represents the largest size, the smallest element in $Sizes$ is selected instead. This procedure is repeated until a termination criterion is satisfied, in the proposed implementation this is that a maximal number of solutions is generated.

Algorithm 4 Pseudocode for the Fixed Set Search

```
Input: Problem instance $A$, Size of RCL $\alpha$, $n$ number of best solutions used for randomly selecting the base solution, $k$ specifies the set of test solutions $\mathcal{S}_{kn}$, control parameter for the stagnation in the FSS $MaxStag$, number of initial runs of the GRASP $N$,parameter for specifying the sizes of the fixed sets $\beta$
Output: Best found soultion $S_{best}$
function FSS(A, $\alpha$, $n$, $k$, $MaxStag$,$N$,$\beta$)
    Initialize $Sizes$ using $\beta$
    $Size = Sizes.Next$
    $\mathcal{S} = GRASP(A, \alpha, N)$
    while (Maximal number of solutions generated) do
        Set $\mathcal{S}_{kn}$ to random $k$ elements of $\mathcal{S}_n$
        Set $B$ to a random solution in $\mathcal{S}_n$
        $F = Fix(B, \mathcal{S}_{kn}, Size|B|)$
        $S = RandomizedGreedyWithFix(F, A, \alpha)$
        $S = LocalSearch(S)$
        $\mathcal{S} = \mathcal{S} \cup \{S\}$
        Check if $S$ is new best $S_{best}$
        if No new solution added to $S_n$ in $M$ iterations then
            $Size = Sizes.Next$
        end if
    end while
end function
```

VII. Results

This section presents the outcomes of the computational experiments conducted to assess the effectiveness of the FSS. A comparison is made between the proposed FSS and GRASP algorithms to the state-of-the-art methods for the MKCP. To be exact, the comparison is done to the best known solutions (BKS) acquired using methods presented in [8]–[10]. In addition, a detailed comparison to the RNKC [7] is provided since it has a high level of similarity to the GRASP but with an improved local search. The FSS and the GRASP methods are implemented in C# using Microsoft Visual Studio Community 2022. The computational experiments were performed on a personal computer running Windows 10, equipped with an Intel(R) Xeon(R) Gold 6244 CPU operating at 3.60 GHz and 128 GB of memory.

The following parameters have been used for all the test instances for the GRASP and the FSS. These values have been selected empirically. The value $\alpha = 5$ is used for the size of the RCL. The FSS is considered stagnating if $M = 5$ iterations have been performed without adding a solution to the set $\mathcal{S}_n$ of best n solutions. One of the difference to the application of FSS on other problems [11]–[15] is in the way the size of the fixed sets are generated. To be more precise, in the other implementations the value of β is not considered as a parameter and the value of 0.5 is used. In case of the MCKP, this value did not produce the best results instead the value of $\beta = 0.8$ is used. One reason for this could be that the number of potentially selected columns is drastically lower than the

TABLE I
COMPARISON OF THE RKNC, GRASP AND FSS METHODS TO BKS FOR LARGE-SCALE INSTANCES. FOR ALL THE METHODS THE DIFFERENCE TO THE BKS IS PRESENTED FOR THE BEST FOUND AND AVERAGE SOLUTION OVER 10 RUNS.

Instance	K90							K95						
	BKS	Best			Averageg			BKS	Best			Average		
		RNKC	GRASP	FSS	RNKC	GRASP	FSS	BKS	RNKC	GRASP	FSS	RNKC	GRASP	FSS
scpa1	289	1	2	1	2.2	2.0	1.0	295	2	1	1	2.8	1.9	1.2
scpa2	289	1	1	0	2.4	1.0	0.9	295	2	1	1	2.8	1.2	1.0
scpa3	289	1	0	0	1.9	0.4	0.0	295	1	1	0	2.2	1.8	0.0
scpa4	288	1	2	0	1.2	2.0	0.0	294	1	1	0	1.7	1.5	0.0
scpa5	290	1	1	0	1.6	1.5	0.0	296	2	2	1	2.4	2.0	1.0
scpb1	296	0	2	0	1.0	2.0	0.0	299	0	1	0	1.4	1.5	0.0
scpb2	295	0	0	0	0.7	0.5	0.0	299	1	1	1	1.5	1.2	1.0
scpb3	296	1	1	0	2.5	1.0	0.9	299	2	1	1	2.5	1.0	1.0
scpb4	295	1	2	1	2.0	2.0	1.0	299	1	2	1	2.4	2.0	1.4
scpb5	295	0	0	0	0.8	0.8	0.0	298	0	1	0	0.9	1.0	0.4
scpc1	391	3	2	0	3.7	2.1	0.8	397	3	2	1	4.4	2.7	1.0
scpc2	391	3	2	1	4.0	2.5	1.6	397	3	2	1	4.2	2.8	1.5
scpc3	391	3	1	2	4.1	2.6	2.0	397	3	3	1	4.4	3.2	1.9
scpc4	392	4	2	1	4.6	3.2	1.2	397	3	2	0	4.0	2.9	0.4
scpc5	391	4	2	1	4.5	2.7	1.3	397	3	2	2	4.6	2.5	2.0
scpd1	395	4	4	1	4.1	4.0	1.9	398	2	3	0	2.9	3.0	0.9
scpd2	393	1	1	1	2.2	1.0	1.0	397	2	1	1	2.6	1.0	1.0
scpd3	393	2	1	0	3.4	1.5	1.6	397	3	1	0	3.4	1.9	1.2
scpd4	393	3	3	1	4.0	3.0	1.0	397	3	2	1	3.9	2.4	1.1
scpd5	393	3	3	1	3.7	3.0	1.0	397	3	2	1	3.7	2.0	1.4
scpnre1	488	3	2	0	3.3	2.0	0.9	495	2	1	0	3.5	1.0	0.9
scpnre2	488	1	2	1	2.9	2.0	1.0	495	3	2	1	3.3	2.0	1.0
scpnre3	488	1	2	1	3.3	2.0	1.0	495	3	2	0	3.5	2.0	0.9
scpnre4	488	1	2	1	2.4	2.0	1.0	495	3	2	1	3.8	2.0	1.0
scpnre5	488	1	2	1	2.9	2.0	1.0	495	3	2	1	3.7	2.0	1.1
scpnrf1	498	2	3	1	3.0	3.0	1.7	500	0	0	0	0.7	0.0	0.0
scpnrf2	497	2	2	0	2.5	2.0	0.7	500	0	0	0	0.4	0.0	0.0
scpnrf3	498	3	3	2	3.6	3.0	2.0	500	0	0	0	0.6	0.0	0.0
scpnrf4	497	2	2	0	2.5	2.0	0.7	500	0	0	0	0.6	0.0	0.0
scpnrf5	497	1	2	0	2.3	2.0	0.7	500	0	0	0	0.2	0.0	0.0
scpnrg1	984	9	5	1	13.8	6.1	3.5	995	11	4	0	13.1	4.6	2.2
scpnrg2	984	13	5	1	14.9	6.1	2.0	995	13	4	1	13.8	5.5	2.5
scpnrg3	983	12	5	3	14.1	6.1	3.5	993	10	3	2	11.4	3.6	2.5
scpnrg4	982	11	5	1	13.4	5.8	3.2	995	12	6	4	13.6	6.5	4.6
scpnrg5	983	13	5	3	14.2	6.7	4.7	993	11	3	1	11.7	3.9	2.8
scpnrh1	991	9	0	0	10.0	1.1	1.0	996	9	1	1	9.7	1.9	1.2
scpnrh2	992	9	2	1	10.9	2.5	1.6	995	7	0	-1	8.5	0.7	0.1
scpnrh3	993	10	3	2	12.0	3.4	2.7	996	8	1	1	9.4	1.9	1.3
scpnrh4	992	10	2	0	11.0	2.3	1.3	996	7	1	0	8.7	1.9	0.7
scpnrh5	992	9	2	-1	10.5	2.1	1.5	995	7	0	0	8.6	0.9	0.3
scpclr13	4045	0	0	0	1.2	0.0	0.0	4063	8	7	0	9.3	7.5	5.8
scpcyc09	4349	110	89	29	113.7	91.8	31.3	4468	115	93	20	121.8	99.4	24.9
scpcyc10	11004	302	257	41	316.8	263.2	68.2	11292	336	288	72	346.7	295.7	76.2
scpcyc11	26649	736	602	440	748.2	608.1	452.6	27315	769	641	495	790.3	655.1	524.8
$100\frac{BKS-Method}{BKS}$		0.73	0.54	0.19	0.96	0.60	0.28		0.73	0.46	0.19	0.92	0.54	0.26

number of available ones and no similar properties exists in the previously addressed problems. The population size n had the value of 200 for all the instances except the *scpnrcyc* ones. For these instances the selected value is 20 since due to the high density of the graphs a much lower number of solutions could be generated. Similar low values have been used in case of the application of the FSS to clique partitioning problem due to the same reason [14]. The number of selected test solutions ($\mathcal{S}_{kn}$) used for generating the fixed set, k is equal to 5. The number of iterations of the GRASP for generating the initial population N was equal to the population size n. The termination criterion for the FSS and the GRASP is that 30 000 solutions have been generated or a time limit of 120 seconds has been reached.

The tests are performed on the same set of test instances as in [7]–[10], for which a detailed description can be found in [8]. Note that this benchmark test set is a part of the OR-Library. Since, the learning mechanism of the FSS becomes relevant for improving the GRASP for large problem sizes only instances having 3000 or more columns are considered, in total 44. For each of the problem size two values for the number of used columns are evaluated. To be exact, these values are equal to 90% (K90) and 95% (K95) of the number of used columns in the solution of the corresponding set covering problem. Since, the FSS and GRASP are stochastic methods the evaluation is done over 10 independent runs for each instance, as has previously been done for the RKNC. The comparison is done based on the quality of the best found solution and the average solution value. The values for the BKS and the RKNC are collected from the corresponding articles. The results of the computation experiments can be seen in Table I. It should be noted that the times needed to find the best solutions for the GRASP and the FSS are similar to the ones for the RKNC and are not analysed in detail in this paper.

The first observation that can be made from these results is that the GRASP has a better performance than the RKNC. To be exact, the GRASP has higher quality best solutions than RKNC for 21 and 30 instance compared to RKNC being better in 10 and 4 instances for problem types K90 and K95, respectively. When the average solution quality is compared, the advantage of GRASP becomes even more significant, with GRASP being better for 39 and 41 instances for problem types K90 and K95, respectively. The advantage of GRASP is related to the fact that the RKNC uses initial solutions for the local search that are highly random and of much lower quality than the initial solutions used in the GRASP which are acquired using the randomized greedy algorithm.

The use of the learning mechanism of the FSS results in a performance significantly better than the GRASP. When the average solution quality is considered the FSS is better for 86 instance than the GRASP and worse for only one. A similar behavior can be observed for the best found solutions where the FSS had a lower quality solution than the GRASP in only one instance while being better for 71 instances. The advantage of the FSS compared to the RKNC and GRASP is also confirmed when the average relative distance to the BKS is observed. To be exact, the FSS lowers this value by close to 75% and 60% for the RKNC and the GRASP, respectively.

VIII. Conclusion

This paper presents an innovative solution for addressing the MKCP. Specifically, it firstly introduces a GRASP algorithm for the problem of interest. Later this algorithm is extended the FSS. The performed computational experiments shown that the FSS algorithm is highly competitive with state-of-the-art methods. It is important to point out that even though the FSS use a simple local search it managed to find to new best solutions for the standardly used test instances. Furthermore, the conducted tests have clearly indicated that the FSS's learning mechanism consistently delivers a significant improvements when compared to the underlying GRASP algorithm.

The presented research opens up exciting avenues for further exploration. For example, applying the FSS to different variations of the set covering problem. Another direction is the use of more advanced local searches to further improve the performance of the algorithm on the MKCP.

References

[1] M. R. Garey and D. S. Johnson, *Computers and intractability*. freeman San Francisco, 1979, vol. 174.

[2] J.-F. Cordeau, F. Furini, and I. Ljubić, "Benders decomposition for very large scale partial set covering and maximal covering location problems," *European Journal of Operational Research*, vol. 275, no. 3, pp. 882–896, 2019.

[3] Z. Wang, J. Liao, Q. Cao, H. Qi, and Z. Wang, "Achieving k-barrier coverage in hybrid directional sensor networks," *IEEE Transactions on Mobile Computing*, vol. 13, no. 7, pp. 1443–1455, 2013.

[4] Y. Kong, M. Zhang, and D. Ye, "A belief propagation-based method for task allocation in open and dynamic cloud environments," *Knowledge-Based Systems*, vol. 115, pp. 123–132, 2017.

[5] M. Yaghini, M. Karimi, and M. Rahbar, "A set covering approach for multi-depot train driver scheduling," *Journal of Combinatorial Optimization*, vol. 29, pp. 636–654, 2015.

[6] H. Yu and D. Yuan, "Set coverage problems in a one-pass data stream," in *Proceedings of the 2013 SIAM international conference on data mining*. SIAM, 2013, pp. 758–766.

[7] Y. Wang, D. Ouyang, M. Yin, L. Zhang, and Y. Zhang, "A restart local search algorithm for solving maximum set k-covering problem," *Neural Computing and Applications*, vol. 29, pp. 755–765, 2018.

[8] G. Lin and J. Guan, "Solving maximum set k-covering problem by an adaptive binary particle swarm optimization method," *Knowledge-Based Systems*, vol. 142, pp. 95–107, 2018.

[9] G. Lin, H. Xu, X. Chen, and J. Guan, "An effective binary artificial bee colony algorithm for maximum set k-covering problem," *Expert Systems with Applications*, vol. 161, p. 113717, 2020.

[10] Y. Zhou, X. Liu, S. Hu, Y. Wang, and M. Yin, "Combining max–min ant system with effective local search for solving the maximum set k-covering problem," *Knowledge-Based Systems*, vol. 239, p. 108000, 2022.

[11] R. Jovanovic, M. Tuba, and S. Voß, "Fixed set search applied to the traveling salesman problem," in *International Workshop on Hybrid Metaheuristics*. Springer, 2019, pp. 63–77.

[12] R. Jovanovic and S. Voss, "The fixed set search applied to the power dominating set problem," *Expert Systems*, vol. 37, no. 6, p. e12559, 2020.

[13] R. Jovanovic and S. Voß, "Fixed set search application for minimizing the makespan on unrelated parallel machines with sequence-dependent setup times," *Applied Soft Computing*, vol. 110, p. 107521, 2021.

[14] R. Jovanovic, A. P. Sanfilippo, and S. Voß, "Fixed set search applied to the clique partitioning problem," *European Journal of Operational Research*, vol. 309, no. 1, pp. 65–81, 2023.

[15] R. Jovanovic and S. Voß, "Fixed set search applied to the minimum weighted vertex cover problem," in *International Symposium on Experimental Algorithms*. Springer, 2019, pp. 490–504.

[16] I. Lozano-Osorio, J. Sánchez-Oro, A. Martínez-Gavara, A. D. López-Sánchez, and A. Duarte, "An efficient fixed set search for the covering location with interconnected facilities problem," in *Metaheuristics: 14th International Conference*. Springer, 2023, pp. 485–490.

[17] R. Jovanovic, A. P. Sanfilippo, and S. Voß, "Fixed set search applied to the multi-objective minimum weighted vertex cover problem," *Journal of Heuristics*, vol. 28, pp. 481–508, 2022.

[18] T. A. Feo and M. G. Resende, "Greedy randomized adaptive search procedures," *Journal of Global Optimization*, vol. 6, no. 2, pp. 109–133, 1995.