Fixed Set Search Applied to the Minimum Weighted Vertex Cover Problem

Raka Jovanovic[1] and Stefan Voß[2,3(✉)]

[1] Qatar Environment and Energy Research Institute (QEERI), Hamad bin Khalifa University, PO Box 5825, Doha, Qatar
rjovanovic@hbku.edu.qa
[2] Institute of Information Systems, University of Hamburg, Von-Melle-Park 5, 20146 Hamburg, Germany
stefan.voss@uni-hamburg.de
[3] Escuela de Ingenieria Industrial, Pontificia Universidad Católica de Valparaíso, Valparaíso, Chile

Abstract. Fixed set search (FSS) is a novel metaheuristic adding a learning mechanism to enhanced greedy approaches. In this paper we use FSS for solving the Minimum Weighted Vertex Cover Problem (MWVCP). First we define a Greedy Randomized Adaptive Search Procedure (GRASP) by randomizing the standard greedy constructive algorithm and combine it with a local search. The used local search is based on a simple downhill procedure. It checks if substituting a single or a pair of elements from a solution with ones that need to be added to keep the solution a vertex cover decreases the value of the objective function. The performance of the GRASP algorithm is improved by extending it towards FSS. Computational experiments performed on standard test instances from literature show that the proposed FSS algorithm for the MWVCP is highly competitive with state-of-the-art methods. Further, it is shown that the FSS manages to significantly improve the GRASP algorithm it is based on.

Keywords: Metaheuristics · Minimum Weighted Vertex Cover Problem · GRASP · Fixed set search

1 Introduction

The Minimum Vertex Cover Problem (MVCP) is one of the standard combinatorial optimization problems that has been extensively researched. The decision version of the MVCP is one of Karp's 21 NP-complete problems [9]. It is defined for a graph $G(V, E)$ having a set of vertices V and a set of edges E. A vertex set $C \subset V$ is called a vertex cover if for every edge $\{u, v\} \in E$ at least one of the vertices u or v is an element of C. The objective of the MVCP is to find a vertex cover C that has minimum cardinality. In this paper, we focus on solving the Minimum Weighted Vertex Cover Problem (MWVCP) which is a variation of the MVCP in which for each node $u \in V$ there is a corresponding weight w_u.

I. Kotsireas et al. (Eds.): SEA^2 2019, LNCS 11544, pp. 490–504, 2019.
https://doi.org/10.1007/978-3-030-34029-2_31

The objective in the MWVCP is to find the vertex cover having the minimum total weight. Formally, the objective is to find a set $C \subset V$ which minimizes:

$$\sum_{u \in V} w_u x_u \tag{1}$$

In (1), variables of type x_u are equal to 1 if $u \in C$ and zero otherwise. The variables x_u need to satisfy the following constraints:

$$x_u + x_v \geq 1 \qquad (\{u, v\} \in E) \tag{2}$$

The MWVCP well represents a large number of real-world problems related to wireless communication, circuit design, and network flow [11,15] which resulted in an extensive amount of research dedicated to finding optimal and near optimal solutions. It should be noted that the vast majority of research solves the MWVCP with positive coefficients. It has been shown that it can be solved as fast as the unweighted vertex cover in $O(1.2738^p + pN_V)$, with exponential memory use [3,4] (here N_V is the size of the vertex set and p the size of the prospective cover, if it exists). Due to the NP-Hardness of the MWVCP a wide range of methods have been developed for finding near optimal solutions ranging from greedy algorithms to different types of metaheuristics.

In [12], an ant colony optimization (ACO) is presented. The performance of the ACO method has been further improved using a pheromone correction strategy as presented in [7]. The problem has also been solved using genetic algorithms combined with a greedy heuristic [13], a population-based iterated greedy (PBIG) algorithm [1] and a reactive tabu search hybridized with simulated annealing [14]. One of the most successful approaches is the multi-start iterated tabu search (MS-ITS) algorithm [16]. The most successful methods incorporate some types of local searches [10]. In this paper a dynamic scoring strategy is incorporated to improve the local search performance, which produces a computationally very effective method being capable to solve problem instances having hundreds of thousands of nodes and edges. Another method designed to solve problem instances on massive graphs can be found in [2], in which first an initial vertex cover is generated that is later improved using an advanced local search.

Due to the fact that the use of local searches has proven very efficient in case of the MWVCP, in this paper the potential effectiveness of the Greedy Randomized Adaptive Search Procedure (GRASP) [5] is explored. To be more precise, our objective is to see the effectiveness of combining a simple to implement greedy algorithm and local search. Two local searches are presented based on a downhill procedure using swap operations which remove one or two vertices from the solutions and add necessary vertices. The basic idea of the swap operations is very similar to the ones used in [10,14]. The performance of the proposed GRASP algorithm is further improved by extending it to the novel Fixed Set Search metaheuristic [8], which has previously been successfully applied to the Traveling Salesman Problem (TSP). The FSS uses a simple approach to add a learning mechanism to GRASP based on elements frequently appearing in high quality solutions. The performed computational experiments show that the FSS is highly competitive with the state-of-the-art methods in the quality of found

solutions. Further, we show that the FSS produces a significant improvement when compared to the GRASP algorithm on which it is based.

The paper is organized as follows. In Sect. 2, we give a brief description of the randomized greedy algorithm for the MWVCP. In the following section details of the local searches are presented. In Sect. 4, an outline of the GRASP metaheuristic is given. In the next section, we give details of the FSS and how it is applied to the MWVCP. In Sect. 6, we discuss the performed computational experiments. The paper is finalized with concluding remarks.

2 Greedy Algorithm

In this section, the standard greedy constructive algorithm for the MWVCP is presented. The basic idea of the method is to start from a partial solution $S = \emptyset$ and at each step expand it with the vertex that has the most desirable properties based on a heuristic function h. To be more precise, it is preferable to expand the solution with a vertex u that covers the largest number of non-covered edges and has the minimal weight w_u. Formally, the heuristic function for a partial solution S and a node n has the following form.

$$Cov(n, S) = \{\{n, v\} \mid (\{n, v\} \in E) \wedge (v \notin S)\} \tag{3}$$

$$h(n, S) = \frac{|Cov(n, S)|}{w_n} \tag{4}$$

In (3), $Cov(n, S)$ is the set of edges in E that contain node n but are not already covered by S. An edge is covered if at least on of its vertices is in S. The heuristic function, defined in (4), is proportional to the number of elements of $Cov(S, n)$, and reversely proportional to the weight w_n of the vertex n.

Since, our goal is to use the presented 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) as follows. Let us define R as the set of N elements from $v \in V \setminus S$ that have the largest value of $h(v, S)$. Now, we can expand the partial solution with a random element of set R. The pseudocode for the proposed randomized greedy constructive algorithm (RGC) can be seen in Algorithm 1. In it, the partial solution S is initially set to an empty set. At each iteration S is expanded with a random element from the RCL. This is repeated until all the edges E are covered. The proposed RGC has computational complexity of $|S||V|$, where S is the generated solution.

3 Local Searches

In this section, two local searches based on a correction procedure are presented. The basic idea of the proposed local searches is based on the concept of swapping elements of a solution S with elements of $V \setminus S$ that produce a vertex cover but decrease the objective function. This approach has proven to be very successful on the closely related dominating set problem.

Algorithm 1. Pseudocode for the RGC for the MWVCP

```
S = ∅
while Not all edges covered do
    Generate RCL based on h and S
    Select random element n ∈ RCL
    S = S ∪ n
end while
```

3.1 Element Swap

Assume that we aim to improve a solution S. Since S is a vertex cover of G, for each edge $\{u, v\}$ at least one of u or v is an element of S. Let us define $Un(v, S)$, for a solution S and vertex $v \in S$, as the set of vertices that correspond to edges that are uniquely covered by vertex vS as

$$Un(v, S) = \{u \mid u \notin S \wedge \{u, v\} \in E\} \tag{5}$$

It is evident that if we swap a vertex v with all the elements $Un(v, S)$ a new vertex cover will be created. For simplicity of notation let us define the swap operation for a vertex v as

$$Swap(v, S) = (S \cup Un(v, S)) \setminus \{v\} \tag{6}$$

Now, a swap operation for a vertex v can be evaluated as

$$EvSwap(v, S) = w_v - \sum_{i \in Un(v,S)} w_i \tag{7}$$

In Eq. (7), $EvSwap(v)$ gives the change in the solution weight when $v \in S$ is swapped. More precisely, it is equal to the weight w_v of vertex v that is removed from the solution minus the total sum of weights of vertices that are added to the solution. Now, we can define $Imp(S)$ as the set of all vertices of S for which a swap operation produces an improvement

$$Imp(S) = \{v \mid v \in S \wedge EvSwap(v, S) > 0\} \tag{8}$$

3.2 Pair Swap

The basic idea of the swap operation can be extended to pairs of vertices. In case of a local search based on swap pairs it generally is the case that the computational cost will increase $|S|$ times, where S is the solution being improved. Although this cannot be changed asymptotically, it can be greatly decreased in practical applications. It is important to note that in case of the MWVCP swap operations of this type are more effective than for other problems since the elements that are used for substitution are uniquely defined. In designing the local search based on swap pairs, we focus on two objectives. Firstly, to have a

very small overlap with a local search based on element swaps and secondly to increase computational efficacy.

In our application, we assume that in a pair swap operation involving $\{u, v\}$ both elements will be removed and none of them will be re-added. In case this constraint is not used the same effect can be achieved using an element swap. In case such a constraint exists, if $\{u, v\} \in E$ such a pair can never be swapped since the edge $\{u, v\}$ will not be covered. Additional positive effects of a pair swap $\{u, v\}$ can only occur if u and v have overlapping neighborhoods, or in other words in case there is a node w that is adjacent to both u and v. Using this idea, let us formally define the set of improving swap pairs for a solution S. Based on the previous discussion the set of all vertex pairs that should be tested for a graph G can be defined as follows:

$$C_p = \{\{u, v\} \mid (u, v \in V) \wedge (N(v) \cap N(u) \neq \emptyset)\} \setminus E \tag{9}$$

In (9), the notation $N(v)$ is used for the open neighborhood of v, i.e., all nodes adjacent to v not including itself. Using this set of candidate swap pairs for graph G, we can define the set of improving swap pairs in a similar way as a set of improving elements using the following set of equations.

$$Un(u, v, S) = Un(u, S) \cup Un(v, S) \tag{10}$$

$$Swap(u, v, S) = (S \cup Un(u, v, S)) \setminus \{u, v\} \tag{11}$$

$$EvSwap(u, v, S) = w_u + w_v - \sum_{i \in Un(u,v,S)} w_i \tag{12}$$

$$ImpPair(S) = \{\{u, v\} \mid (\{u, v\} \in C_p \cap S^2) \wedge (EvSwap(u, v, S) > 0)\} \tag{13}$$

In (10), $Un(u, v, S)$ corresponds to the set of nodes that correspond to the set of edges that are uniquely covered by one of the vertices u or v. Note, that this set excludes vertices u and v. In (11), the effect of the swapping elements u and v from a solution S is given. To be more precise, the vertices u and v are removed from the solution S, and all the nodes corresponding to uniquely covered edges are added. $EvSwap(u, v, S)$, given in (12), is equal to the change on the weight of the solution if the vertex pair $\{u, v\}$ is swapped. Finally, in the next equation $ImpPair(S)$ is the set of all swap pairs that improve the quality of the solution from the set of the restricted list of candidate pairs. The restricted set of candidate pairs is equal to the intersection of the unordered product of set S with itself S^2 and the set of all candidate pairs for graph G.

3.3 Local Search

In this subsection, we present the local search based on the presented improvement using element swaps and pair swaps. It should be noted that these two types of improvement explore different neighborhoods of a solution S. Because of this, as in the case of the variable neighborhood search [6], it is advantageous

Algorithm 2. Pseudocode for the local search based on swap operations

```
repeat
    while Imp(S) ≠ ∅ do
        Select random v ∈ Imp(S)
        S = Swap(v, S)
    end while
    if ImpPair(S) ≠ ∅ then
        Select random {v, u} ∈ ImpPair(S)
        S = Swap(u, v, s)
    end if
until (Imp(S) = ∅) ∧ (ImpPair(S) = ∅)
```

to use both of them interchangeably. The pseudocode for the local search based on swap operations can be seen in Algorithm 2.

In Algorithm 2, a solution S is interchangeably improved based on swap elements and swap pairs. Firstly, all the possible improvements are performed using swap elements since this operation is computationally less expensive. This is done by repeatedly performing swap element improvements until no further improvement of this type is possible. Next, we test if an improvement can be achieved using swap pairs. If this is true, the improvement is performed. As there is a possibility, that after applying a swap pair improvement new element swaps can produce improvement, the main loop is repeated until no such improvement exists. It should be noted that for both types of improvements there are several different ways to select the swap that will be performed; in the proposed implementation we simply select a random one.

4 GRASP

To enhance the performance of the proposed greedy algorithm and local search, we extend them to the GRASP metaheuristic as illustrated in Algorithm 3. In the main loop of Algorithm 3, a new solution S to the MWVCP is generated using the RGC algorithm. The local search is applied to the solution S and tested if it is the new best solution. This procedure is repeated until some stopping criterion is satisfied, usually a time limit or a maximal allowed number of solutions has been generated.

Algorithm 3. Pseudocode for the GRASP

```
while Not Stop Criteria Satisfied do
    Generate Solutions S using randomized greedy algorithm
    Apply local search to S
    Check if S is the new best
end while
```

5 Fixed Set Search

The fixed set search (FSS) is a novel metaheuristic that adds a learning mechanism to the GRASP. Literally it uses elite solutions, consistent solution elements or alike to direct the search. It has previously been successfully applied to the TSP [8]. The FSS has several important positive traits. Firstly, there is a wide range of problems on which it can possibly be applied (this paper tries to put evidence on it) since the only requirement is that the solution of the problem is represented in a form of a set. The learning mechanism is simple to implement and many existing GRASP algorithms can easily be extended to this form. In this section the general concepts used in the FSS are presented as well as details of its application to the MWVCP. A more detailed explanation of the concepts used in the FSS can be found in [8].

The main inspiration for the FSS is the fact that generally many high quality solutions for a combinatorial optimization problem contain some common elements. The idea is to use such elements to steer the search of the solution space. To be more precise, we wish to force such elements in a newly generated solution and dedicate computational effort to finding optimal or near optimal solutions in the corresponding subset of the solution space. The selected set of common elements will be called the *fixed set*. In the FSS, we are trying to find the additional elements to complete the partial solution, corresponding to the fixed set, or in other words to "fill in the gaps." In practice, we are intensifying the search around such fixed sets. This can be achieved through the following steps. Firstly, a method for generating fixed sets needs to be implemented. Next, the randomized greedy algorithm used in the corresponding GRASP needs to be adapted in a way to be able to use a preselected set of elements. Lastly, the learning mechanism which gains experience from previously generated solutions needs to be specified.

5.1 Fixed Set

Let us first define a method that will make it possible to generate random fixed sets. As previously stated the FSS can be applied to a problem for which a solution can be represented in a form of a set S having elements in some set W, or in other words $S \subset W$. In case of the MWVCP this concerns a solution $S \subset V$. In the following the notation $\mathcal{P}$ will be used for the set of all the generated solutions (population). Next, let us define $\mathcal{P}_n \subset \mathcal{P}$ as the set of n solutions having the best value of the objective function inside $\mathcal{P}$.

One of the requirements of the FSS is that the method used to generate a fixed set F has the ability to control its size $|F|$. Further, such fixed sets need to be able to produce high quality feasible solutions. This can be achieved using a base solution $B \in \mathcal{P}_m$. If the fixed set satisfies $F \subset B$, it can be used to generate the base solution. In practice this means it can generate a feasible solution at least of the same quality as B, and F can contain arbitrary elements of B. It is preferable for F to contain elements that frequently occur in some group of high quality solutions. To achieve this, let us define $\mathcal{S}_{kn}$ as the set of k randomly selected solutions out of the n best ones P_n.

Using these building blocks it is possible to define a function $Fix(B, \mathcal{S}_{kn}, Size)$ for generating a fixed set $F \subset B$ that consists of $Size$ elements of the base solution $B = \{v_1, ...v_l\}$ that most frequently occur in $\mathcal{S}_{kn} = \{S_1, .., S_k\}$. Let use define the function $C(v_x, S)$, for an element $v_x \in V$ and a solution $S \subset V$, which is equal to 1 if $v_x \in S$ and 0 otherwise. We can define a function that counts the number of occurrences of element v_x in the elements of the set $\mathcal{S}_{kn}$ using the function $C(v_x, S)$ as follows.

$$O(v_x, \mathcal{S}_{kn}) = \sum_{S \in \mathcal{S}_{kn}} C(v_x, S) \tag{14}$$

Now, we can define $Fix(B, \mathcal{S}_{kn}, Size)$ as the set of $Size$ elements $v_x \in B$ that have the largest value of $O(v_x, \mathcal{S}_{kn})$.

5.2 Learning Mechanism

The learning mechanism in the FSS is implemented through the use of fixed sets. To achieve this it is necessary to adapt the RGC algorithm used in the corresponding GRASP to a setting where some elements are preselected (the newly generated solution must contain them). Let us use the notation $RGF(F)$ for the solution generated using such an algorithm with a preselected (fixed) set of elements F. In case of the MWVCP, the RGC algorithm is trivially adapted to a $RGF(F)$ by setting the initial partial solution S to F instead of an empty set.

In the FSS, as in the case of the GRASP, solutions are repeatedly generated and a local search is applied to each of them. The first step is generating an initial population of solutions $\mathcal{P}$ by performing N iterations of the corresponding GRASP algorithm. The initial population is used to generate a random fixed set F having some size $Size$, using the method from the previous section. The fixed set F is used to generate a new solution $S = RGF(F)$ and the local search is applied to it. The population of solutions is expanded using the newly generated locally optimal solutions. This procedure is repeated until no new best solutions are found for a long period by some criteria, or in other words until stagnation has occurred. In case of stagnation the size of the fixed set is increased. In case the maximal allowed size of the fixed set is reached, the size of the fixed set is reset to the minimal allowed value. This procedure is repeated until some stopping criterion is reached. An important part of the algorithm is defining the array of allowed fixed set sizes, which is related to the part of the solution that is fixed. In our implementation this array is defined as follows:

$$Sizes[i] = (1 - \frac{1}{2^i}) \tag{15}$$

The size of the used fixed sets is proportional to the used base solution B. More precisely, at the i-th level it is equal to $|B| \cdot Size[i]$.

The pseudocode for FSS can be seen in Algorithm 4. In it, the first step is initializing the sizes of fixed sets using (15). The current size of the fixed

Algorithm 4. Pseudocode for the Fixed Set Search

```
Initialize Sizes
Size = Sizes.Next
Generate initial population P using GRASP(N)
while (Not termination condition) do
    Set S_kn to random k elements of P_n
    Set B to a random solution in P_m
    F = Fix(B, S_kn, Size|B|)
    S = RGF(F)
    Apply local search to S
    P = P ∪ {S}
    if Stagnant Best Solution then
        Size = Sizes.Next
    end if
end while
```

set $Size$ is set to the smallest value. The next part of the initialization stage is generating the initial population of N solutions by performing N iterations of the basic GRASP algorithm. Each iteration of the main loop consists of the following steps. A random set of solutions $\mathcal{S}_{kn}$ is generated by selecting k elements from $\mathcal{P}_n$ and a random base solution B is selected from the set $\mathcal{P}_m$. Next, the function $Fix(B, \mathcal{S}_{kn}, Size|B|)$ is used to generate a fixed set F. A new solution $S = RGF(F)$ is generated using the randomized greedy algorithm with preselected elements and the local search is applied to it. Next, we check if S is the new best solution and add it to the set of generated solutions $\mathcal{P}$. In case stagnation has occurred, the value of $Size$ is set to the next value in $Sizes$. Let us note, that the next size is the next larger element of array $Sizes$. In case $Size$ is already the largest size, we select the smallest element in $Sizes$. This procedure is repeated until some termination criterion is satisfied.

In our implementation of the proposed algorithm for the MWVCP, the criterion for stagnation was that no new best solution has been found in the last, say, $Stag$ iterations. As previously stated the adaptation of the randomized constructive greedy algorithm to the $RGF(F)$ consists of simply setting the initial partial solution to the fixed set instead of an empty set. The set of candidate swap pairs C_p is calculated in the initialization stage. At this time the set of all neighboring vertices for valid candidate pairs $\{u, v\}$ are also calculated with the intention of speeding the calculation of $ImpPair(S)$.

6 Results

In this section we give details of the performed computational experiments. Their objective is to evaluate the performance of the proposed GRASP and FSS in combination with the element- (GRASP-E and FSS-E) and pair- (GRASP-P and FSS-P) based local searches. Note that the pair-based local search is only

used in combination with the element-based one. This has been done in comparison with the ACO algorithm from [12] and its improvement version (ACO-SEE) [7]. Further, a comparison is with the population-based iterated greedy (PBIG) algorithm [1], the multi-start iterated tabu search (MS-ITS) algorithm [16] and the Diversion Local Search based on Weighted Configuration Checking (DLSWC) [10] which are the best performing methods. Note that the reactive tabu search hybrid produces about the same quality of results than DLSWC, but in [14] results are only presented for a small subset of instances.

The comparison is done on the set of test instances introduced in [12], that have been also used to evaluate the other mentioned methods. The test instances are divided into three groups: small, medium and large. In case of the small and medium test instances, random graphs having 10–300 nodes and 10–5000 edges are used for evaluation. For each pair (N_V, N_E) with N_V and N_E being the number of vertices and edges, respectively, there are ten different graph instances. The test instances are divided into Type 1 where there is no correlation between the weight of a vertex and number on incident edges, and Type 2 where some weak correlation exists; details can be found in [12]. In case of large test instances the graphs have between 500 and 1000 vertices and between 500 and 20 000 edges, and there is only one instance for each pair (N_V, N_E).

The used parameters for FSS are the following, $k = 10$ random solutions are selected from the best $n = 100$ ones for the set of solutions $\mathcal{S}_{kn}$. The base solution is selected from the $m = 100$ best solutions. The size of the initial population is 100. The stagnation criterion is that no new best solution is found in the last $Stag = 100$ iterations for the current fixed set size. The used size of the RCL in the randomized greedy algorithm is 10. The stopping criterion for all the proposed methods is that 5000 solutions are generated or a time limit of 10 minutes has been reached. The FSS and GRASP have been implemented in C# using Microsoft Visual Studio 2017. The calculations have been done on a machine with Intel(R) Core(TM) i7-2630 QM CPU 2.00 Ghz, 4 GB of DDR3-1333 RAM, running on Microsoft Windows 7 Home Premium 64-bit.

In Tables 1 and 2 the results for the medium-size problem instances are given for graphs of Type 1 and Type 2, respectively. For each pair (N_V, N_E), the average weight of all the vertex covers of this type are evaluated. With the intention of having a clearer presentation, the average value of the objective function is only given for DLSWC, while for the other methods only the difference to this value is presented. The values for the methods used for comparison are taken from the corresponding papers. Note that we did not include the results for small problem instances since all the methods except the two ACO methods manage to find all the optimal solutions. From the results in these tables it can be seen that the two ACO algorithms have a substantially worse performance than the other methods. Further, the FSS-P had the overall best performance of all the methods except DLSWC, having on average only 0.8 and 0.1 higher value of the objective function. It should be noted that although FSS overall has a worse performance than DLSWC in case of two pairs (N_V, N_E) it managed to find higher quality average solutions. The experiments performed on large test

Table 1. Comparison of the methods for medium-size problem instances of Type 1.

$N_V \times N_E$	ACO		Element		Pair		PBIG	MS-ITS	DLSWC
	Basic	SEE	GRASP	FSS	GRASP	FSS			
50 × 50	2.1	0.9	0.0	0.0	0.0	0.0	0.0	0.0	1280.0
50 × 100	5.8	5.4	0.0	0.0	0.0	0.0	0.0	0.0	1735.3
50 × 250	15.1	8.3	0.0	0.0	0.0	0.0	0.0	0.0	2272.3
50 × 500	17.1	7.4	0.0	0.0	0.0	0.0	0.0	0.0	2661.9
50 × 750	8.0	6.3	0.0	0.0	0.0	0.0	0.0	0.0	2951.0
50 × 1000	17.5	6.1	0.0	0.0	0.0	0.0	0.0	0.0	3193.7
100 × 100	18.7	9.8	0.0	0.0	0.0	0.0	3.4	0.0	2534.2
100 × 250	24.8	13.3	0.0	0.0	0.0	0.0	1.1	0.0	3601.6
100 × 500	91.5	35.8	0.0	0.0	0.0	0.0	0.0	0.0	4600.6
100 × 750	30.9	37.3	0.0	0.0	0.0	0.0	0.0	0.0	5045.5
100 × 1000	25.9	14.5	0.0	0.0	0.0	0.0	1.2	0.0	5508.2
100 × 2000	43.8	16.4	0.0	0.0	0.0	0.0	0.0	0.0	6051.9
150 × 150	18.0	9.9	1.4	0.4	0.9	0.0	0.4	0.1	3666.9
150 × 250	49.8	35.0	1.8	0.0	0.0	0.0	0.4	0.0	4719.9
150 × 500	58.6	63.3	6.7	0.0	0.0	0.0	0.3	0.0	6165.4
150 × 750	58.3	39.9	7.6	6.8	0.0	0.0	7.3	10.6	6956.4
150 × 1000	82.1	23.9	8.0	1.6	1.6	1.6	9.1	0.0	7359.7
150 × 2000	81.8	47.8	0.0	0.2	0.0	0.0	12.6	0.0	8549.4
150 × 3000	50.4	40.4	0.0	0.0	0.0	0.0	0.0	0.0	8899.8
200 × 250	37.1	20.8	3.6	0.0	0.0	0.0	0.3	0.0	5551.6
200 × 500	67.3	41.8	2.6	1.1	0.0	0.0	0.5	3.2	7191.9
200 × 750	79.9	30.4	6.6	2.5	1.2	1.2	4.6	0.0	8269.9
200 × 1000	116.7	62.9	23.6	4.5	5.3	1.8	5.1	4.5	9145.5
200 × 2000	86.5	61.1	10.8	3.9	0.3	0.4	1.0	0.0	10830.0
200 × 3000	93.3	84.6	0.2	0.0	0.0	0.0	4.4	3.8	11595.8
250 × 250	49.1	20.5	4.6	0.0	0.0	0.0	0.0	0.0	6148.7
250 × 500	102.6	59.7	20.9	7.0	6.7	3.1	4.5	2.6	8436.2
250 × 750	123.5	69.6	24.8	6.4	1.9	-0.3	6.9	0.0	9745.9
250 × 1000	114.9	39.3	14.7	1.1	1.7	0.0	2.0	0.4	10751.7
250 × 2000	166.2	75.5	25.8	3.1	3.2	2.2	6.1	4.4	12751.5
250 × 3000	159.2	107.3	23.2	6.4	0.0	0.0	0.2	0.0	13723.3
250 × 5000	132.1	66.2	8.0	0.0	0.0	0.0	7.0	0.0	14669.7
300 × 300	46.9	30.8	7.0	0.2	0.0	0.0	0.2	0.0	7295.8
300 × 500	114.3	88.8	55.1	11.4	8.3	7.7	0.0	7.7	9403.1
300 × 750	137.6	127.2	60.9	14.0	13.3	2.2	8.8	2.7	11029.3
300 × 1000	143.2	65.2	40.2	8.9	4.6	4.4	10.4	9.2	12098.5
300 × 2000	162.7	102.4	49.7	15.4	6.7	2.7	17.7	5.5	14732.2
300 × 3000	213.3	69.7	42.3	1.7	1.0	1.0	7.4	0.6	15840.8
300 × 5000	202.5	136.9	31.6	19.3	2.3	1.9	7.7	0.0	17342.9
Average	78.18	45.70	12.35	2.97	1.51	0.77	3.35	1.42	
Found Best	0	0	14	19	24	27	12	25	

Table 2. Comparison of the methods for medium-size problem instances of Type 2.

$N_V \times N_E$	ACO		Element		Pair		PBIG	MS-ITS	DLSWC
	Basic	SEE	GRASP	FSS	GRASP	FSS			
50 × 50	0.2	0.2	0.0	0.0	0.0	0.0	0.0	0.0	83.7
50 × 100	5.0	3.2	0.0	0.0	0.0	0.0	0.0	0.0	271.2
50 × 250	33.4	16.9	0.0	0.0	0.0	0.0	0.0	0.0	1853.4
50 × 500	90.8	51.6	0.0	0.0	0.0	0.0	0.0	0.0	7825.1
50 × 750	55.1	8.6	0.0	0.0	0.0	0.0	0.0	0.0	20079.0
100 × 50	0.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	67.2
100 × 100	2.5	1.2	0.0	0.0	0.0	0.0	0.0	0.0	166.6
100 × 250	15.2	8.8	0.0	0.0	0.0	0.0	0.0	0.0	886.5
100 × 500	33.1	13.4	0.0	0.0	0.0	0.0	0.0	0.0	3693.6
100 × 750	74.3	62.1	0.0	0.0	0.0	0.0	0.0	0.0	8680.2
150 × 50	0.0	0.1	0.0	0.0	0.0	0.0	0.0	0.0	65.8
150 × 100	0.7	0.1	0.0	0.0	0.0	0.0	0.0	0.0	144.0
150 × 250	9.9	9.0	0.2	0.0	0.0	0.0	0.2	0.0	615.8
150 × 500	43.5	27.1	1.1	0.0	0.0	0.0	0.0	0.0	2331.5
150 × 750	100.7	8.5	0.9	0.0	0.0	0.0	0.2	0.0	5698.5
200 × 50	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	59.6
200 × 100	0.2	0.1	0.0	0.0	0.0	0.0	0.0	0.0	134.5
200 × 250	5.6	4.8	0.3	0.0	0.0	0.0	0.0	1.4	483.1
200 × 500	39.7	14.8	0.2	0.0	0.0	0.1	0.4	0.0	1803.9
200 × 750	69.3	33.5	0.0	0.2	0.2	0.2	0.1	0.0	4043.5
250 × 250	4.2	2.2	0.0	0.0	0.0	0.0	0.0	0.0	419.0
250 × 500	23.2	20.1	1.9	0.4	0.5	0.0	1.5	0.5	1434.2
250 × 750	59.8	33.3	3.1	0.0	0.0	0.0	4.9	0.3	3256.1
250 × 1000	71.8	53.6	7.7	−0.1	−0.3	−0.3	3.0	1.8	5986.4
250 × 2000	512.6	295.6	42.1	6.2	0.0	0.0	22.0	9.9	25636.5
250 × 5000	1648.2	1231.7	120.2	28.9	0.1	0.1	0.1	0.1	170269.0
300 × 250	4.5	3.3	0.4	0.0	0.1	0.0	0.1	0.2	399.4
300 × 500	22.7	20.9	2.1	0.0	0.2	0.0	0.0	0.8	1216.4
300 × 750	38.9	34.8	4.5	0.6	0.4	0.0	0.1	1.3	2639.3
300 × 1000	100.5	72.9	17.2	1.8	6.6	1.8	1.3	1.2	4795.0
300 × 2000	413.9	226.4	60.6	6.6	0.0	2.5	10.3	5.1	20881.3
300 × 5000	2023.1	1072.2	109.9	11.5	4.8	4.8	44.9	6.4	141220.4
Average	171.96	104.09	11.64	1.75	0.39	0.29	2.78	0.91	
Found Best	2	2	16	23	24	26	18	20	

Table 3. Comparison of best found solutions over 10 independent runs for large problem instances by different methods.

$N_V \times N_E$	ACO	Element		Pair		PBIG	MS-ITS	DLSWC
	SEE	GRASP	FSS	GRASP	FSS			
500 × 500	59.0	43.0	5.0	5.0	0.0	0.0	7.0	12616.0
500 × 1000	51.0	36.0	2.0	1.0	0.0	0.0	15.0	16465.0
500 × 2000	137.0	204.0	10.0	0.0	0.0	0.0	0.0	20863.0
500 × 5000	53.0	387.0	0.0	21.0	0.0	77.0	0.0	27241.0
500 × 10000	0.0	165.0	0.0	0.0	0.0	0.0	0.0	29573.0
800 × 500	24.0	44.0	0.0	0.0	0.0	0.0	21.0	15025.0
800 × 1000	45.0	99.0	0.0	15.0	0.0	0.0	13.0	22747.0
800 × 2000	379.0	472.0	144.0	102.0	16.0	54.0	8.0	31301.0
800 × 5000	277.0	518.0	159.0	163.0	62.0	112.0	0.0	38553.0
800 × 10000	148.0	290.0	45.0	41.0	6.0	45.0	0.0	44351.0
1000 × 1000	133.0	288.0	36.0	34.0	9.0	23.0	12.0	24723.0
1000 × 5000	243.0	460.0	62.0	79.0	61.0	52.0	27.0	45203.0
1000 × 10000	497.0	742.0	61.0	92.0	0.0	0.0	0.0	51378.0
1000 × 15000	400.0	670.0	133.0	169.0	61.0	20.0	20.0	57994.0
1000 × 20000	359.0	523.0	128.0	84.0	27.0	139.0	24.0	59651.0
Average	187.00	329.40	52.33	53.73	16.13	34.80	9.80	
Found Best	1	0	4	3	8	7	6	

Table 4. Comparison of average quality of found solutions over 10 independent runs for large problem instances by different methods.

$N_V \times N_E$	ACO	Element		Pair		PBIG	MS-ITS	DLSWC
	SEE	GRASP	FSS	GRASP	FSS			
500 × 500	71.7	68.0	5.0	5.0	2.0	4.0	19.0	12616.0
500 × 1000	109.9	56.8	5.6	2.0	0.0	5.1	18.1	16465.0
500 × 2000	226.8	226.4	12.2	−3.2	−3.2	4.6	0.7	20866.2
500 × 5000	344.5	411.0	91.2	64.8	0.0	187.2	0.0	27241.0
500 × 10000	223.4	199.4	86.8	3.0	0.0	93.8	0.0	29573.0
800 × 500	44.9	53.6	0.0	0.0	0.0	0.0	29.1	15025.0
800 × 1000	105.1	119.4	0.0	19.0	0.0	16.0	13.0	22747.0
800 × 2000	481.9	502.6	227.4	157.2	17.6	117.6	40.7	31305.0
800 × 5000	337.6	561.7	149.3	171.9	72.5	149.6	−12.0	38569.1
800 × 10000	337.8	350.9	42.1	51.1	15.1	43.9	6.0	44353.9
1000 × 1000	202.4	315.6	45.8	34.0	29.0	40.1	43.1	24723.0
1000 × 5000	349.8	469.3	73.9	92.3	25.3	56.5	18.0	45238.9
1000 × 10000	724.6	755.2	88.4	128.0	−2.4	160.5	42.6	51380.4
1000 × 15000	659.8	723.8	132.0	223.4	77.0	150.2	73.9	57995.0
1000 × 20000	612.9	597.3	148.7	149.5	80.3	192.6	64.6	59655.3
Average	322.21	360.73	73.89	73.20	20.88	81.45	23.79	

instances can be seen in Tables 3 and 4 where the values of the best found and average weight over 10 runs of each algorithm are given, respectively. In case of problems instances of this size a similar behavior can be seen as for medium-size instances.

From the computational results it is evident that the methods FSS-P and GRASP-P in which the local search includes pair swaps manages to find significantly better solutions than the element-based ones. The use of pair swaps in the local search produces a more significant improvement than the addition of the learning mechanism used in the FSS. It should be noted that GRASP-P has a better performance than FSS-E for medium-size instances, while FSS-E manages to have a slightly better performance for large instances with FSS-P being consistently better in both cases. The improvement that is achieved by FSS is more significant in case of the weaker local search based on element swaps. However, it is most important to note that the improvement achieved by FSS compared to the corresponding GRASP is very consistent, and it only has worse average quality of found solutions for 1 or 3 of the (N_V, N_E) pairs, when the used local search was based on elements or pairs, respectively.

The convergence speed of the FSS-P is competitive to other methods, in case of medium-size instances it needs an average time of 0.76 and 0.43 s to find the best solution for Type 1 and Type 2 graphs, respectively. This is a very similar result to MS-ITS which needed between 0.51 and 0.45 s to solve instances of Type 1 and Type 2, and better than PBIG which needs 2.49 and 4.23 s. DLSWC has a substantially better performances; on average it needs only 0.03 and 0.4 s. The FSS-P scales well, and for large graphs needs an average of 5.05 s to find the best solution for an instance which is similar to 5.20 of DLWSC, but it should be noted that the quality of solutions is of lower quality. The scaling of PBIG and MS-ITS is significantly worse and the methods on average need 126.94 and 74.80 s to solve large problem instances, respectively. It is interesting to point out that although the asymptotic computational cost of the local search based on pairs is greater than the one based on elements, the time for finding the best solutions for GRASP-P and FSS-P is generally 2–5 times lower than for GRASP-E and FSS-E. The FSS, on average, needs around half the time of GRASP with the same type of local search to find the best solution. The pair swap local search proves to be very efficient; for graphs having up to 100 nodes GRASP-P generally needs less than 20 iterations to find the best known solutions.

7 Conclusion

In this paper we have presented an efficient easy to implement method for finding near optimal solutions for the MWVCP. This has been done by developing two local searches based on a correction procedures which switches one or two vertices from a solution with new ones which produces a new vertex cover having a lower weight. These local searches have been used as a part of a GRASP algorithm. The performance of the developed GRASP has been improved by extending it to the novel Fixed Set Search metaheuristic. The conducted computational

experiments have shown that the proposed FSS is highly competitive with the state-of-the-art methods. The results also indicate that the learning mechanism in the FSS manages to significantly enhance its performance when compared to the GRASP on which it is based. Importantly, the positive effect is most significant on large-scale problem instances on which the effectiveness of GRASP algorithms is generally decreased. For future research we aim to extend the application of the FSS to other types of problems.

References

1. Bouamama, S., Blum, C., Boukerram, A.: A population-based iterated greedy algorithm for the minimum weight vertex cover problem. Appl. Soft Comput. **12**(6), 1632–1639 (2012)
2. Cai, S., Li, Y., Hou, W., Wang, H.: Towards faster local search for minimum weight vertex cover on massive graphs. Inf. Sci. **471**, 64–79 (2019)
3. Chen, J., Kanj, I.A., Xia, G.: Improved upper bounds for vertex cover. Theor. Comput. Sci. **411**(40–42), 3736–3756 (2010)
4. Cygan, M., Kowalik, Ł., Wykurz, M.: Exponential-time approximation of weighted set cover. Inf. Process. Lett. **109**(16), 957–961 (2009)
5. Feo, T.A., Resende, M.G.: Greedy randomized adaptive search procedures. J. Glob. Optim. **6**(2), 109–133 (1995). https://doi.org/10.1007/BF01096763
6. Hansen, P., Mladenović, N.: Variable neighborhood search: principles and applications. Eur. J. Oper. Res. **130**(3), 449–467 (2001)
7. Jovanovic, R., Tuba, M.: An ant colony optimization algorithm with improved pheromone correction strategy for the minimum weight vertex cover problem. Appl. Soft Comput. **11**(8), 5360–5366 (2011)
8. Jovanovic, R., Tuba, M., Voß, S.: Fixed set search applied to the traveling salesman problem. In: Blesa Aguilera, M.J., Blum, C., Gambini Santos, H., Pinacho-Davidson, P., Godoy del Campo, J. (eds.) HM 2019. LNCS, vol. 11299, pp. 63–77. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-05983-5_5
9. Karp, R.M.: Reducibility among combinatorial problems. In: Miller, R.E., Thatcher, J.W., Bohlinger, J.D. (eds.) Complexity of computer computations, pp. 85–103. Springer, Boston (1972). https://doi.org/10.1007/978-1-4684-2001-2_9
10. Li, R., Hu, S., Zhang, H., Yin, M.: An efficient local search framework for the minimum weighted vertex cover problem. Inf. Sci. **372**, 428–445 (2016)
11. Pullan, W.: Optimisation of unweighted/weighted maximum independent sets and minimum vertex covers. Discrete Optim. **6**(2), 214–219 (2009)
12. Shyu, S.J., Yin, P.Y., Lin, B.M.: An ant colony optimization algorithm for the minimum weight vertex cover problem. Ann. Oper. Res. **131**(1–4), 283–304 (2004). https://doi.org/10.1023/B:ANOR.0000039523.95673.33
13. Singh, A., Gupta, A.K.: A hybrid heuristic for the minimum weight vertex cover problem. Asia-Pac. J. Oper. Res. **23**(02), 273–285 (2006)
14. Voß, S., Fink, A.: A hybridized tabu search approach for the minimum weight vertex cover problem. J. Heuristics **18**(6), 869–876 (2012)
15. Wang, L., Du, W., Zhang, Z., Zhang, X.: A PTAS for minimum weighted connected vertex cover P3 problem in 3-dimensional wireless sensor networks. J. Comb. Optim. **33**(1), 106–122 (2017)
16. Zhou, T., Lü, Z., Wang, Y., Ding, J., Peng, B.: Multi-start iterated tabu search for the minimum weight vertex cover problem. J. Comb. Optim. **32**(2), 368–384 (2016)