Fixed Set Search Applied to the Max-Cut Problem

Irina Šević
Faculty of Mathematics
University of Belgrade
Belgrade, Serbia
isevic.37@gmail.com

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

Dragan Urošević, Tatjana Davidović
Mathematical Institute,
Serbian Academy of Science and Arts
Belgrade, Serbia
{draganu,tanjad}@turing.mi.sanu.ac.rs

***Abstract*—The Max-Cut Problem (MCP) is a classical NP-hard combinatorial optimization problem for graph partitioning, which has many applications such as optimizing electrical grids, or wireless sensor networks. In the context of this paper, the fixed set search (FSS) which is novel metaheuristic that uses a population-based approach is applied for solving the MCP. Initially, the greedy randomized adaptive search procedure (GRASP) is formulated to address the given problem. Subsequently, the FSS incorporates a learning procedure into the GRASP by identifying common elements within high-quality solutions. The primary benefit of this metaheuristic is the simplicity of implementation. The algorithms are tested on standard benchmark instances. The conducted computational experiments validate that the learning procedure of the FSS enhances the effectiveness of the base GRASP method, and outperforms other population based metaheuristics that include local search procedures like the AntCut and the hierarchical social metaheuristics.**

***Index Terms*—Graph partitioning, population-based metaheuristics, learning mechanisms, optimizing electrical microgrids, optimizing wireless sensor network**

I. Introduction

The Max-Cut Problem (MCP) is an optimization challenge that involves partitioning the nodes of a graph into two sets to maximize the number of edges crossing between the two sets. It has a wide range of applications like VLSI circuit design [1], image segmentation [2] and statistical physics [3]. In the recent years, it has been applied for optimization of electrical grids, since they can be modeled as a graphs with nodes representing power generation points, such as wind turbines and solar panels, and edges representing transmission lines. The MCP can be used to determine the optimal way to partition the network. For instance, the MCP has proven highly effective in modeling and optimizing microgrids, in the presence of distributed energy resources [4] or the integration of charging infrastructure for electric vehicles (EVs) [5]. Another important application of the MCP is optimizing wireless sensor networks [6], [7] which can be of high relevance to smartgrid monitoring.

The MCP is among the most investigated combinatorial optimization problems and is one of Karp's 21 NP-complete problems [8]. Due to the NP-hardness of the MCP, many heuristic and metaheuristic methods have been created for finding near optimal solutions. In [9], a randomized greedy heuristic and swap based local search have been proposed. In the same paper, this basic approach is extend to the greedy randomized adaptive search procedure (GRASP), variable neighborhood search (VNS) and combined with path-relinking. In [10] an overview of several greedy heuristics is given. It is important to mention that an extensive analysis of the effectiveness of such heuristics for the MCP is provided in [11].

The MCP has also been addressed using single solution metaheuristics like simulated annealing and tabu search [12], [13]. Metaheuristics integrating population-based approaches alongside local searches have demonstrated remarkable effectiveness in addressing the MCP, like ant colony optimization [14], artificial bee colony algorithm [15] and approaches based on the genetic algorithms [16]–[18]. Other population based metaheuristics have also proven effective for the MCP, e.g. particle swarm optimization [19], scatter search [20], harmony search [21], etc. From the extensive research on the MCP, it is notable that it has been used in developing approaches based on neural networks [22] and graph neural networks [23].

The Fixed Set Search (FSS) has found successful application across various combinatorial problems including the traveling salesman problem [24], machine scheduling [25], clique partitioning problem [26] and others. It has also been successfully extend to bi-objective problems [27] and a matheuristic setting [28]. The FSS is a population-based metaheuristic with an integrated local search, that is particularly suitable for solving the MCP, given the success of local search-based methods. FSS enhances the GRASP by adding a learning procedure, which focuses on elements that commonly arise in high-quality solutions, termed the "fixed set". This approach aims to generate solutions that incorporate these elements, with computational effort directed at completing partial solutions. In this paper, we explore the application of the FSS to the MCP. The performed computational experiments suggest that the FSS competes effectively with other metaheuristics incorporating a local search component.

The structure of the paper is outlined as follows: Section II contains the formulation of the MCP. The next section is dedicated to solving the MCP using the greedy algorithm (Section III). The following two section are focused on the local search (Section IV) and the GRASP algorithm (Section V). Section VI provides details of the FSS algorithm. Section VII presents the outcomes of the carried out computational

979-8-3503-8215-0/24/$31.00 ©2024 IEEE

experiments, along with their analysis. Finally, the paper ends with concluding remarks and references.

II. Formulation of the Max-Cut Problem

The formal definition of MCP can be stated as follows: Consider an undirected graph $G = (V, E)$, where V is a set of vertices and E is a set of edges. In addition, each edge $(i, j) \in E$, $i, j \in V$, has an assigned integer weight w_{ij}. The goal of the MCP is to divide the set of vertices into two subsets, S and $S^c = V \setminus S$ for which the sum of the weights of the edges with one vertex in S and the other in S^c is maximized. This set of edges is commonly called the cut, and is fully specified by the set S. To formally, specify the MCP, let us first define the function $w(S_1, S_2)$ for two vertex sets S_1 and S_2.

$$w(S_1, S_2) = \sum_{(i,j)\in E | i\in S_1, j\in S_2} w_{ij} \tag{1}$$

In Eq. (1) the $w(S_1, S_2)$ is equal to the sum of all weights of all edges (i, j) where $i \in S_1$ and $j \in S_2$. To simplify the notation, let's introduce the function $w(S) = w(S, S^c)$, for a set of vertices $S \subset V$. The objective of MCP is to find the subset of vertices S that maximizes the value of $w(S)$. A graphical illustration of the MCP can be seen in Fig. 1.

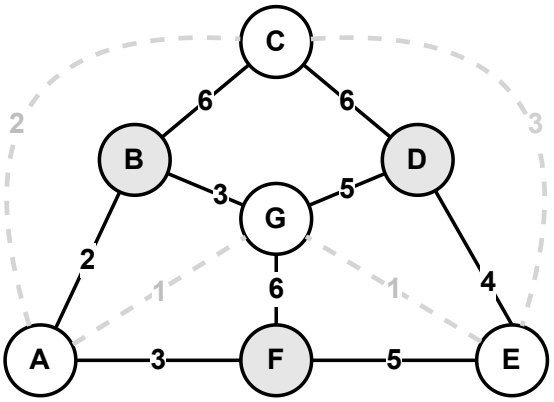

Fig. 1. Depiction of a solution to a MCP instance. The nodes of the two subsets S and S^c are colored white and gray, respectively. The values on the edges represent edge weights. The black color is used for edges that are included in the cut (edges having one node in S and the other in S^c), while gray dashed ones are not included in the cut. The objective function's value is $w(S) = 34$.

III. Greedy constructive algorithm

The concept behind the greedy algorithm involves starting with an initial partial solution $\mathcal{S} = (S_1, S_2)$. Let this partial solution consist of sets $S_1 = \{u\}$ and $S_2 = \{v\}$, where the edge $(u, v) \in E$ has the maximum weight w_{uv}. This partial solution is iteratively expand by adding a vertex $v \in V \setminus (S_1 \cup S_2)$ to one of the disjoint sets S_1 or S_2 until $S_1 \cup S_2 = V$. In relation, let us define $V' = V \setminus (S_1 \cup S_2)$ as the set of vertices that are not yet included in the partial solutions (S_1, S_2). At this point, we can define the list of candidates for expanding the partial solution (S_1, S_2) using the following equation.

$$C(S_1, S_2) = \{(v, S) \mid v \in V' \wedge S \in \{S_1, S_2\}\} \tag{2}$$

The list of candidates consists of pairs (v, S), where $v \in V'$ and $S \in \{S_1, S_2\}$, with the meaning that vertex v is added to vertex set S. To simplify the notation in the further text, let us define the following function

$$d(S) = \begin{cases} S_1 & S = S_2 \\ S_2 & S = S_1 \end{cases} \tag{3}$$

At each iteration of the greedy algorithm, the partial solution (S_1, S_2) is extended by incorporating the candidate (v, S), resulting in a new partial solution with the maximum weight for the cut. Consequently, the heuristic function used in the greedy algorithm for a candidate (v, S) can be defined as follows.

$$h(v, S) = \sum_{u\in d(S)} w_{uv} \tag{4}$$

Eq.(4) simply states that the desirability of adding vertex v to vertex set S equals the sum of weights across all edges connecting vertex v to the vertices within $d(S)$. Now, in the greedy algorithm at each iteration, the candidate (v, S) having the maximal value of $h(v, S)$ is selected for expanding the partial solution.

To integrate the greedy algorithm into the GRASP meta-heuristic, it must include randomization. This is commonly accomplished by employing a restricted candidate list (RCL) approach. The idea of the RCL is that, in each iteration of the greedy algorithm, the partial solution is not expanded with the candidate corresponding to the largest value of the heuristic function. Instead, one of the candidates, identified as good enough according to the RCL, is selected randomly.

In the case of the MCP, randomization is introduced through the following procedure. We start with a partial solution $S_1 = \{u\}$ and $S_2 = \{v\}$, where the edge $(u, v) \in E$ is one of the edges with the maximum weight.

Algorithm 1 Construction of greedy randomized solution

```
1:  procedure GreedyRandomizedConstruction(α, K)
2:      u, v ← SelectRandomMaxEdge(K)
3:      S_1 ← {u}, S_2 ← {v}
4:      while V ≠ S_1 ∪ S_2 do
5:          μ ← CalculateThreashold(S_1, S_2, w_*, w^*, α)
6:          RCL ← MakeRCL(μ)
7:          Set (v, S) to random element of RCL
8:          if S = S_1 then
9:              S_1 ← S_1 ∪ {v}
10:         else
11:             S_2 ← S_2 ∪ {v}
12:         end if
13:     end while
14: return S_1, S_2
15: end procedure
```

The RCL is generated using the method proposed in [29], which is briefly recalled here. Let us define RCL as a list of all candidates (v, S) that have the heuristic value $h(v, S)$ greater than or equal to a threshold $\mu = w_* + \alpha(w^* - w_*)$, where α is a

random value uniformly selected from the $[0, 1]$ interval and w_* and w^* are defined as follows:

$$w_* = \min_{(v,S)\in C(S_1,S_2)} h(v, S) \quad (5)$$

$$w^* = \max_{(v,S)\in C(S_1,S_2)} h(v, S) \quad (6)$$

In practice this means, that w_* and w^* are equal to the minimal and maximal value of the heuristic function of all the candidates for expansion, respectively.

The proposed randomized greedy algorithm is best understood by observing the pseudo-code given in Alg. 1. The first step in Alg. 1 is to find an initial cut by randomly selecting an edge (u, v) amongst the K with the maximal weight. In the main loop, we start with updating the value of the threshold μ for the RCL using the function CalculateThreashold. Next, the RCL is generated based on μ, a random candidate (v, S) is selected, and it is used to expand the partial solution. This process is repeated until all vertices are allocated to one of the subsets. Note that the randomized greedy algorithm uses α as a parameter that specifies the level of randomness.

IV. Local search

The concept behind the local search is to start from a candidate solution (S_1, S_2) and see if moving a vertex v from its current subset S to the other subset $d(S)$ results in a solution of higher quality. To formally specify this procedure, let us define the function Move(S_1, S_2, v), for solution (S_1, S_2) and vertex v that returns the new solution (S'_1, S'_2) where the vertex v is moved from the current subset S to the subset $d(S)$. To be exact,

- $S'_1 = S_1 \setminus \{v\}$, $S'_2 = S_2 \cup \{v\}$, if $v \in S_1$
- $S'_2 = S_2 \setminus \{v\}$, $S'_1 = S_1 \cup \{v\}$, if $v \in S_2$.

Next, let us define the function $Imp(S_1, S_2)$ for a solution (S_1, S_2) that returns the set of all vertices $v \in V$ such that $w(\text{Move}(S_1, S_2, v))$ is greater than $w(S_1, S_2)$. In Alg. 2 the pseudo-code for the proposed local search procedure is shown using this function. Instead of an exhaustive search over all elements identified by the function $Imp(S_1, S_2)$, a random vertex is selected, the current solution is improved by its movement from one subset to the other, and $Imp(S_1, S_2)$ is invoked to update the set of vertices that lead to the new improvements. This process is iterated until no additional improvement can be attained.

Algorithm 2 Local search

```
1: procedure LocalSearch(S_1, S_2)
2:     while Imp(S_1, S_2) ≠ ∅ do
3:         Set v to random element of Imp(S_1, S_2)
4:         (S_1, S_2) ← Move(S_1, S_2, v)
5:     end while
6: return S_1, S_2
7: end procedure
```

V. GRASP

The GRASP metaheuristic operates by iteratively employing the randomized greedy algorithm to generate solutions, followed by the application of a local search to refine each of them. This is best understood by observing the pseudo-code given in Alg. 3. Within the main loop, the initial step is randomly selecting the value of variable α that is used to specify the level of randomness of the greedy construction algorithm in that iteration. The next step is generating a feasible solution Sol using the function GreedyRandomizedConstruction(α, K). After that the local search is applied to that solution using the function LocalSearch(Sol). The final step is checking if a new best solution has been acquired. This procedure is repeated until a maximal number of iterations (*maxitr*) has been reached.

Algorithm 3 GRASP

```
1: procedure GRASP(maxitr, K)
2:     BestSol ← ∅
3:     for k = 1 : maxitr do
4:         α ← Random(0, 1)
5:         Sol ← GreedyRandomizedConstruction(α, K)
6:         Sol ← LocalSearch(Sol)
7:         UpdateSolution(BestSol, Sol)
8:     end for
9: return BestSol
10: end procedure
```

VI. Fixed Set Search

The FSS is an extension of the GRASP metaheuristic, which adds a learning mechanism based on identified common elements in high quality solutions. In the case of the MCP, these elements correspond to pairs (v, i) consisting of vertices $v \in V$ and their partitions $i \in \{1, 2\}$. Such common elements are used to generate new solutions by putting vertices in the corresponding partitions. The FSS involves constructing an initial population, generating fixed sets F, iteratively creating new solutions by employing the learning mechanism that exploits common elements and updating the set of high-quality solutions.

Let us define the following notation. A solution is a set of pairs (v, i) for $v \in V$ and $i \in \{1, 2\}$, where (v, i) means that $v \in S_i$. In this formulation, each solution $Sol \subseteq V \times \{1, 2\}$. Note that this representation is equivalent to the one used in the previous sections. Let $\mathcal{S}$ be the initial population, i.e. a set of all generated solutions. We define $\mathcal{S}_n = \{Sol_1, .., Sol_n\} \subseteq \mathcal{S}$ as the set of the best n generated solutions.

The procedure for generating a fixed set consist of the following steps. A base solution B is selected randomly from $\mathcal{S}_n$. If the fixed set F is a subset of B, it holds the potential to generate a feasible solution of equal or superior quality compared to B. Moreover, F can include any number of elements from B. The concept behind FSS is to incorporate into F those elements that are the most frequent in some group of high-quality solutions $\mathcal{S}_n$. Let $\mathcal{S}_{kn} \subseteq \mathcal{S}_n$ denote the

set of k solutions randomly chosen from $\mathcal{S}_n$. Finally, fixed sets customized for the MCP can be effectively constructed by using the previously defined components.

For an element (v, i) and a solution $Sol \subseteq V \times \{1, 2\}$, let us define the function $C((v, i)), Sol)$ which is equal to 1 if $(v, i) \in Sol$ and equal to 0 otherwise. Let us define the functions $O((v, i), \mathcal{S}_{kn})$ which tallies the number of occurrences of element (v, i) in the solutions within $\mathcal{S}_{kn}$, using the expression:

$$O(v, \mathcal{S}_{kn}) = \sum_{Sol \in \mathcal{S}_{kn}} C((v, i), Sol). \tag{7}$$

Note that, in the implementation of the function $O(v, \mathcal{S}_{kn})$, the symmetry of a solutions $A_1 = (S_1, S_2)$ and $A_2 = (S_2, S_1)$ needs to be addressed. Namely, if $|A_1 \cap B| > |A_2 \cap B|$ and the representation in the form of a set of elements is used, the solution A_1 is used when calculating the function C, and vice versa. In practice, this means that the set A having a higher level of similarity with the base solution B is selected.

As mentioned earlier, fixed sets should contain elements that frequently occur in the set of high-quality solutions. Consequently, a fixed set F is a subset of B consisting of the elements (v, i), where $v \in V$ and $i \in \{1, 2\}$, with the largest values of $O((v, i), \mathcal{S}_{kn})$. Such fixed sets F can be generated with a specified cardinality $Size$. We denote this process as the function F = MakeFixedSet$(B, \mathcal{S}_{kn}, Size)$.

A. Randomized greedy algorithm with pre-selected elements

One of the main components of the FSS is a generation of new solutions containing elements from the fixed set. This can be achieved by the adaptation of the greedy randomized construction algorithm discussed in Section III. Let F be the pre-selected set of elements, i.e. a set of vertices paired with their partition. Instead of initializing the partitions with only two vertices (corresponding to one of the maximum weighted edges), the initial solution begins with the partition of vertices v to i for each $(v, i) \in F$. To distinguish this version from the regular greedy randomized construction algorithm, we denote it as GreedyRandomizedFixed(α, F).

B. Learning mechanism

In this section, we describe the FSS learning mechanism, and how it gains experience from previous solutions. First, we use GRASP for exploring the solution space and generate the initial population $\mathcal{S}$ containing N solutions. We can easily find $\mathcal{S}_n$, $n \leq N$, during this process. Secondly, the FSS iteratively generates solutions, by creating a fixed set F of cardinality $Size$ using past solutions, and generating a new solution Sol using F and the randomized greedy algorithm. Then, the local search improves Sol. Sol is used to update the set $\mathcal{S}_n$ and the best found solution, if it is better than the previous best. This procedure continues iteratively until a stopping criterion is reached.

The fixed set F is produced by utilizing a base solution B and a set of test solutions $\mathcal{S}_{kn}$. We set the the minimum allowed value for it's size, and incrementally increase it after M iterations without finding a new solution good enough to be added to $\mathcal{S}_n$. We avoid repetitive solutions by setting an upper bound on the size of the fixed set $|F|$. The size is reset to its minimum allowed value if it reaches this bound during stagnation. This procedure is repeated until the stopping criterion is reached.

It's crucial to emphasize that it is possible to construct equivalent solutions, i.e. solutions with partitions (S_1, S_2) and (S_2, S_1), which negatively effects the diversity of solution in $\mathcal{S}_n$. To avoid this kind of symmetry, after generating each solution S_1 and S_2 are swapped such that $|S_1| < |S_2|$ holds.

The sizes of the fixed sets within the FSS learning mechanism are determined relative to size of the base solutions. In our implementation, an array of permissible sizes is defined as follows:

$$Sizes[i] = \left\lfloor (1 - \beta^{i+1}) \cdot |V| \right\rfloor \tag{8}$$

where $\beta \in (0, 1)$. The upper limit for the fixed set's size must meet the condition $|V| - Sizes[i] \geq s$ for a predetermined number s. In Alg. 4, the pseudo-code for the implemented Fixed Set Search is shown.

Algorithm 4 Fixed Set Search

```
1:  procedure FSS(α, n, k, M, N, β, s)
2:      Sizes ← InitializeSizes(β,s)
3:      GRASP(N)
4:      i ← 0
5:      while not(stopping criterion) do
6:          B ← SelectRandom(S_n)
7:          S_kn ← SelectRandomK(S_n, k)
8:          F ← MakeFixedSet(B, S_kn, Sizes[i])
9:          Sol ← GreedyRandomizedFixed(α, F)
10:         Sol ← LocalSearch(Sol)
11:         S_n ← UpdateSn(S_n, Sol)
12:         if Sol is better than Sol_best then
13:             Sol_best ← Sol
14:         end if
15:         if S_n has not changed in the last M iterations then
16:             i ← i + 1 (or i ← 0 if i =Size(Sizes))
17:         end if
18:     end while
19: end procedure
```

First, in this algorithm, the sizes of fixed sets are initialized using equation (8) based on input parameters β and s. The size of the current fixed set is represented as $Sizes[i]$. The starting population of solutions $\mathcal{S}$ is generated by executing N iterations of the basic GRASP algorithm before the FSS procedure starts. Iterations of the main loop are executed until a stopping criterion is reached, and each iteration consists of the following steps: first, a base solution B and a set of k solutions $\mathcal{S}_{kn}$ are chosen randomly from the set $\mathcal{S}_n$. Secondly, a fixed set F is generated using the function MakeFixedSet$(B, \mathcal{S}_{kn}, Sizes[i])$. Next, a new solution Sol is created using the randomized greedy algorithm with pre-selected elements GreedyRandomizedFixed(α, F), and improved by local search. Then, the algorithm checks if Sol is

one of the best n solutions and updates $\mathcal{S}_n$ if it is. It also updates the best solution if needed. If stagnation occurs, i.e. no solution is added to $\mathcal{S}_n$ after M iterations, the index i which controls the size of the fixed set is increased. This equals to the subsequent, larger, element in the $Sizes$ array. If $Sizes[i]$ already denotes the final and largest size, i is reset to 0. The used stopping criterion is that either a time limit or a maximal number of iterations is reached.

VII. Results

We compare the implemented FSS and GRASP methods to each other and to the AntCut [14] and the Hierarchical Social (HS) [16] metaheuristics for the MCP, because they are population methods with a local search. Precisely, the comparison is done based on the best known solutions (BKS): the table with results contains the differences between BKS and the solutions acquired by each of the compared methods. The implementation of the FSS and the GRASP is done in C++, compiled with g++ (9.4.0) compiler, and executed on a computer running Ubuntu 20.04.6 LTS with an Intel(R) Celeron(R) CPU 4205U operating at 1.80GHz $\times$ 2 and 120 GB of memory. The FSS and the GRASP have been independently run 10 times with varying seeds for the random number generator. The analysis includes examining both the maximum and average values of the obtained objective function values.

The following values of the parameters, which have been selected empirically, are used. The value α needed for generating the RCL is selected uniformly random from $[0, 1]$. The initial population of solutions $\mathcal{S}$ is created within $N = 100$ GRASP iterations. The FSS stagnation criterion is set to $M = 10$ iterations, while $n = 50$ best solutions are kept as a reference. To generate the set of test solutions $\mathcal{S}_{kn}$, k is randomly selected from the set $\{5, 6, 7, 8, 9, 10\}$ in each iteration. The value 0.5 is used for β, which defines the array of sizes of the fixed set. The stopping condition for both FSS and GRASP is specified as that either 1500 iterations have been performed or a time limit of 360 seconds has been reached.

The experimental results are presented in Table I. The experiments are conducted using the identical set of benchmark instances as in [14], [16], where the results of the AntCut and the HS can be found, respectively. The instances and the BKS values can be found on OR-Library. It is important to note that only the BKS value of instance G3 is taken from [14], as it is larger. In the last three rows, for each of the heuristics (except FSS), the table shows the number of instances that have better, equal or worse results compared to the value computed using FSS.

As depicted in Table I, the FSS's learning mechanism delivers superior results compared to the base GRASP algorithm, considering both maximal and average solution qualities. For all of the instances, it delivers better solutions than at least one of the AntCut and HS heuristics. The FSS found higher quality best solutions for 14 and 15 instances out of 24, while being worse in 6 and 5 than AntCut and HS, respectively. When comparing the average solution quality, the superiority of FSS becomes even more pronounced: it has produced better solutions for 15 and 20 instances compared to AntCut and HS, respectively. The worse performance with respect to the average solution quality is observed only for graphs with higher edge densities.

VIII. Conclusion

We developed the FSS approach to the MCP. The conducted computational experiments have demonstrated that the FSS algorithm outperforms other population based metaheuristics that also incorporate local searches. It has been shown that the FSS learning mechanism manages to significantly improve the performance of the base GRASP method.

The presented research can be extended by using more advanced local searches. Due to the simplicity of the FSS another avenue of research is hybridization with other metaheuristics like simulated annealing. The high effectiveness of the proposed method on the MCP indicates the potential for application on real-world problems corresponding to large problem instances.

Acknowledgement

This research has been partially supported by the Serbian Ministry of Science, Technological Development and Innovations through the Mathematical Institute of the Serbian Academy of Sciences and Arts, Agreement No. 451-03-47/2023-01/200029.

References

[1] K. Chang and D.-C. Du, "Efficient algorithms for layer assignment problem," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 6, no. 1, pp. 67–78, 1987.

[2] S. de Sousa, Y. Haxhimusa, and W. G. Kropatsch, "Estimation of distribution algorithm for the max-cut problem," in *Graph-Based Representations in Pattern Recognition: 9th IAPR-TC-15 International Workshop, GbRPR 2013, Vienna, Austria, May 15-17, 2013. Proceedings 9*. Springer, 2013, pp. 244–253.

[3] F. Barahona, M. Grötschel, M. Jünger, and G. Reinelt, "An application of combinatorial optimization to statistical physics and circuit layout design," *Operations Research*, vol. 36, no. 3, pp. 493–513, 1988.

[4] H. Jing, Y. Wang, Y. Li, L. Du, and Z. Wu, "Quantum approximate optimization algorithm-enabled der disturbance analysis of networked microgrids," in *2022 IEEE Energy Conversion Congress and Exposition (ECCE)*, 2022, pp. 1–5.

[5] ——, "Dynamics analysis of microgrids integrated with ev charging stations based on quantum approximate optimization algorithm," in *2022 IEEE Transportation Electrification Conference and Expo (ITEC)*, 2022, pp. 574–578.

[6] A. Deshpande, S. Khuller, A. Malekian, and M. Toossi, "Energy efficient monitoring in sensor networks," in *LATIN 2008: Theoretical Informatics: 8th Latin American Symposium, Búzios, Brazil, April 7-11, 2008. Proceedings 8*. Springer, 2008, pp. 436–448.

[7] H.-Y. Yang, W.-C. Peng, and C.-H. Lo, "Optimizing multiple in-network aggregate queries in wireless sensor networks," in *International Conference on Database Systems for Advanced Applications*. Springer, 2007, pp. 870–875.

[8] R. M. Karp, *Reducibility among Combinatorial Problems*. Boston, MA: Springer US, 1972, pp. 85–103.

[9] P. Festa, P. M. Pardalos, M. G. Resende, and C. C. Ribeiro, "Randomized heuristics for the max-cut problem," *Optimization methods and software*, vol. 17, no. 6, pp. 1033–1058, 2002.

[10] S. Kahruman, E. Kolotoglu, S. Butenko, and I. V. Hicks, "On greedy construction heuristics for the max-cut problem," *International Journal of Computational Science and Engineering*, vol. 3, no. 3, pp. 211–218, 2007.

TABLE I
DIFFERENCES OF THE BEST KNOWN SOLUTION (BKS) AND ANTCUT, HIERARCHICAL SOCIAL (HS) METAHEURISTIC, GRASP AND FSS, RESPECTIVELY.

Instance				Upper	Best				Average			
Graph	$\|V\|$	Edge Density	BKS	bound	AntCut	HS	GRASP	FSS	AntCut	HS	GRASP	FSS
G1	800	6.12	11624	12078	13	75	109	27	37.5	179.2	127.4	53.5
G2	800	6.12	11620	12084	15	119	88	22	39	173.8	126.8	48.4
G3	800	6.12	11634	12077	0	84	100	29	24.6	191.2	134.2	47.7
G11	800	0.63	564	627	45	18	46	18	54.2	31.5	58	29
G12	800	0.63	556	621	38	16	44	26	48.3	30.4	56.4	30.6
G13	800	0.63	580	645	51	12	42	18	61.8	28.3	58.4	26
G14	800	1.58	3060	3187	18	46	55	18	29.2	73.4	60.6	33
G15	800	1.58	3049	3169	34	56	61	15	40.5	79.6	67.5	29.9
G16	800	1.58	3045	3172	28	49	44	15	36.8	71.5	58.8	26.1
G22	2000	1.05	13346	14123	102	285	280	90	121.5	352.3	329.9	127
G23	2000	1.05	13317	14129	99	219	238	67	118.7	320	310.8	123.8
G24	2000	1.05	13314	14131	158	225	229	42	176.9	323.7	304	98.6
G32	2000	0.25	1398	1560	136	38	136	76	148.2	67.1	153	86.4
G33	2000	0.25	1376	1537	138	52	128	74	154.1	76.1	134.2	86.6
G34	2000	0.25	1372	1541	174	38	130	50	202.9	63.2	155.4	69.6
G35	2000	0.64	7670	8000	124	122	170	81	148.3	182.3	184.2	101.3
G36	2000	0.64	7660	7996	129	136	169	78	149.8	177.8	185.4	91
G37	2000	0.64	7666	8009	96	118	161	59	119.4	177.3	177	83.7
G43	1000	2.1	6659	7027	8	98	99	18	27.5	168.1	128.2	44.8
G44	1000	2.1	6648	7022	8	108	91	23	34.1	159.5	123	46.8
G45	1000	2.1	6652	7020	12	88	103	34	33.4	167.1	130.9	55.2
G48	3000	0.17	6000	*	0	0	0	0	10.3	68.8	0	0
G49	3000	0.17	6000	*	0	0	0	0	8.4	69.3	0	0
G50	3000	0.17	5880	*	0	0	0	0	3.8	39.2	0	0
Average					59.42	83.42	105.13	36.67	76.22	136.28	127.67	55.79
#Better					6	5	0		9	4	0	
#Equal					4	4	3		0	0	3	
#Worse					14	15	21		15	20	21	

[11] I. Dunning, S. Gupta, and J. Silberholz, "What works best when? a systematic evaluation of heuristics for max-cut and qubo," *INFORMS Journal on Computing*, vol. 30, no. 3, pp. 608–624, 2018.

[12] E. Arráiz and O. Olivo, "Competitive simulated annealing and tabu search algorithms for the max-cut problem," in *Proceedings of the 11th annual conference on genetic and evolutionary computation*, 2009, pp. 1797–1798.

[13] G. A. Kochenberger, J.-K. Hao, Z. Lü, H. Wang, and F. Glover, "Solving large scale max cut problems via tabu search," *Journal of Heuristics*, vol. 19, pp. 565–571, 2013.

[14] L. Gao, Y. Zeng, and A. Dong, "An ant colony algorithm for solving max-cut problem," *Progress in Natural Science*, vol. 18, no. 9, pp. 1173–1178, 2008.

[15] X. Chen, G. Lin, and M. Xu, "Applying a binary artificial bee colony algorithm to the max-cut problem," in *2019 12th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI)*. IEEE, 2019, pp. 1–4.

[16] A. Duarte, F. Fernández, Á. Sánchez, and A. Sanz, "A hierarchical social metaheuristic for the max-cut problem," in *European Conference on Evolutionary Computation in Combinatorial Optimization*. Springer, 2004, pp. 84–94.

[17] Q. Wu, Y. Wang, and Z. Lü, "A tabu search based hybrid evolutionary algorithm for the max-cut problem," *Applied Soft Computing*, vol. 34, pp. 827–837, 2015.

[18] S.-H. Kim, Y.-H. Kim, and B.-R. Moon, "A hybrid genetic algorithm for the max cut problem," in *Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation*, 2001, pp. 416–423.

[19] G. Lin and J. Guan, "An integrated method based on PSO and EDA for the max-cut problem," *Computational intelligence and neuroscience*, vol. 2016, pp. 11–11, 2016.

[20] R. Martí, A. Duarte, and M. Laguna, "Advanced scatter search for the max-cut problem," *INFORMS Journal on Computing*, vol. 21, no. 1, pp. 26–38, 2009.

[21] Y.-H. Kim, Y. Yoon, and Z. W. Geem, "A comparison study of harmony search and genetic algorithm for the max-cut problem," *Swarm and evolutionary computation*, vol. 44, pp. 130–135, 2019.

[22] J. Wang, "An improved discrete hopfield neural network for max-cut problems," *Neurocomputing*, vol. 69, no. 13-15, pp. 1665–1669, 2006.

[23] W. Yao, A. S. Bandeira, and S. Villar, "Experimental performance of graph neural networks on random instances of max-cut," in *Wavelets and Sparsity XVIII*, vol. 11138. SPIE, 2019, pp. 242–251.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] R. Jovanovic, S. Bayhan, and S. Voß, "Matheuristic fixed set search applied to electric bus fleet scheduling," in *Learning and Intelligent Optimization*, M. Sellmann and K. Tierney, Eds. Cham: Springer International Publishing, 2023, pp. 393–407.

[29] P. Festa, P. Pardalos, M. Resende, and C. Ribeiro, "Grasp and vns for max-cut," 01 2003.