An Ant Colony Optimization Algorithm for Partitioning Graphs with Supply and Demand

Raka Jovanovic[a], Milan Tuba[b], Stefan Voß[c]

[a] *Qatar Environment and Energy Research Institute, Hamad bin Khalifa University, PO Box 5825, Doha, Qatar*
[b] *Faculty of Computer Science, Megatrend University, Belgrade, Serbia*
[c] *Institute of Information Systems, University of Hamburg, Von-Melle-Park 5, 20146 Hamburg, Germany*
and Escuela de Ingenieria Industrial, Pontificia Universidad Católica de Valparaíso, Chile

Abstract

In this paper we focus on finding high quality solutions for the problem of maximum partitioning of graphs with supply and demand (MPGSD). There is a growing interest for the MPGSD due to its close connection to problems appearing in the field of electrical distribution systems, especially for the optimization of self-adequacy of interconnected microgrids. We propose an ant colony optimization algorithm for the problem. With the goal of further improving the algorithm we combine it with a previously developed correction procedure. In our computational experiments we evaluate the performance of the proposed algorithm on trees, 3-connected graphs, series-parallel graphs and general graphs. The tests show that the method manages to find optimal solutions for more than 50% of the problem instances, and has an average relative error of less than 0.5% when compared to known optimal solutions.

Keywords: Ant Colony Optimization, Microgrid, Graph Partitioning, Demand Vertex, Supply Vertex, Combinatorial Optimization

1. Introduction

In recent years the research in the field of smart grids has had a significant increase in exploring the concept of interconnected microgrids [1]. The main reason for this shift is the increasing participation of Distributed Energy Resources (DER), like wind turbines, solar panels, etc... In practice a microgrid consist of a cluster of DERs and loads which can be, to a certain extent, controlled autonomously form the rest of the grid. It has been shown that by the use of microgrids, Photovoltaics (PV) systems can be effectively included into an existing infrastructure [1]. One of the main advantages of this approach is the lowering of the interaction between the local production and the national electric grid, utilizing only a limited number of connection points. This approach brings significant advantages to the users inside of the microgrid, such as the increased reliability and the independence of the main grid.

This approach has resulted in novel types of typologies for electrical grids and new aspects of such systems that should be considered. Some of the most prominent newly emerged problems are the maximization of self-adequacy [2], reliability, supply-security [3] and the potential for self-healing [4] of such systems. In many cases the underlying optimization problems are of a very high complexity and can not be solved to proven optimality in polynomial time. Electrical grids are systems of gigantic size, which makes their optimization very hard from a computational point of view. Luckily, previous research has shown that for many systems it is not necessary to use highly detailed models; often simplified graph models can give sufficiently good approximations to the original problem. The family of graph partitioning problems has proven to be closely related to power supply and delivery networks [5, 6, 7, 8, 9].

In a system of interconnected microgrids each microgrid is made as independent from the rest of the system as possible; this results in many positive characteristics. Some examples are the lower complexity of the entire grid and enhanced reliability of each of the microgrids due to the increased resistance to failures in other parts of the system. The term independent is used for the case when there is a minimum of power exchange between the connected microgrids. This property of the system is formally defined as the maximization of self-adequacy of interconnected microgrids. Recently, research has been conducted in developing algorithms for finding approximate solutions [2] to this problem. Previous research has also explored the closely related problem of efficient decomposition or islanding of large grids into islands with a balanced generation/load subject to specific constraints [10, 11]. Due to the large complexity and size of electrical grids, when attempting to model and optimize some global properties, it is frequently convenient to use simplified graph models. Such models often result in different versions of graph partitioning problems suitable for specific real life applications. Some examples are having a balanced partitioning [12], minimizing the number or weight of cuts [13, 14], or by limiting the number of cuts [15].

The focus of this paper is on the Maximal Partitioning of Graphs with Supply and Demand (MPGSD). The majority of previous research has been dedicated to the theoretical aspects of this problem [16, 17, 6, 18]. A significant part of the published work is focused on solving this problem for specific types of graphs like trees [17, 6, 18] and series-parallel graphs [16]. A method for finding solutions with a guarantee of a $2k$-approximation for general graphs has been presented in [9]. Different variations of the original problem have been developed, like a parametric version [19] and one with additional capacity constraints [20].

In this paper we present an ant colony optimization (ACO) [21] approach for finding high quality approximate solutions to the MPGSD. ACO has previously been successfully applied to problems of multiway [22] and balanced [23] graph partitioning. The same method has also proven to be suitable for the closely related problems of graph cutting [24] and covering [25, 26] and partitioning of meshes [27]. The proposed ACO adaptation for our problem of interest is based on the greedy algorithm presented in [28, 29]. The ACO algorithm is further improved by combining it with our previously developed correction procedure [29]. In our tests on trees, 3-connected graphs, series-parallel graphs and general graphs, we show that the newly developed method frequently manages to find optimal solutions and has a small average error when compared to known optimal solutions.

The paper is organized as follows. In the second section we give the definition of the MPGSD followed by a section discussing related work. Then we provide a short outline of a greedy algorithm which is used as a basis for the proposed method. Section 5 describes a GRASP algorithm for reasons of comparison. In the sixth section we present details of our ACO algorithm. In the subsequent sections we discuss results of our computational experiments and provide some conclusions.

2. Maximal Partitioning of Graphs With Supply/Demand

The MPGSD is defined for an undirected graph $G = (V, E)$ with a set of nodes V and a set of edges E. The set of nodes V is split into two disjunct subsets V_s and V_d. Each node $u \in V_s$ will be called a supply vertex and will have a corresponding positive integer value $sup(u)$. Elements of the second subset $v \in V_d$ will be called demand vertices and will have a corresponding positive integer value $dem(v)$. The goal is to find a partitioning $\Pi = \{S_1, S_2, .., S_n\}$ of the graph G that satisfies the following constraints. All the subgraphs in Π must be connected subgraphs containing only a single distinct supply node. As a result we have $|V_s| = n$. Each of the S_i must have a supply greater or equal to its total demand. Each demand vertex can be an element of only one subgraph, or in other words it can only receive 'power' from one supply vertex through the edges of G. For simplicity of notation let as assume that the node v_i is the supply node of subgraph S_i.

The goal is to maximize the fulfillment of demands, or more precisely to maximize the following sum.

$$D(\Pi) = \sum_{S \in \Pi} \sum_{v \in S \cap V_d} dem(v) \tag{1}$$

while the following constraints are satisfied for all $S_i \in \Pi$

$$\sum_{v \in S_i \cap V_d} dem(v) \leq \ sup(v_i) \tag{2}$$

$$S_i \cap S_j = \emptyset \quad , \ i \neq j \tag{3}$$

$$S_i \ \ is \ \ connected \tag{4}$$

An illustration of the MPGSD is given in Figure 1. It has been shown that

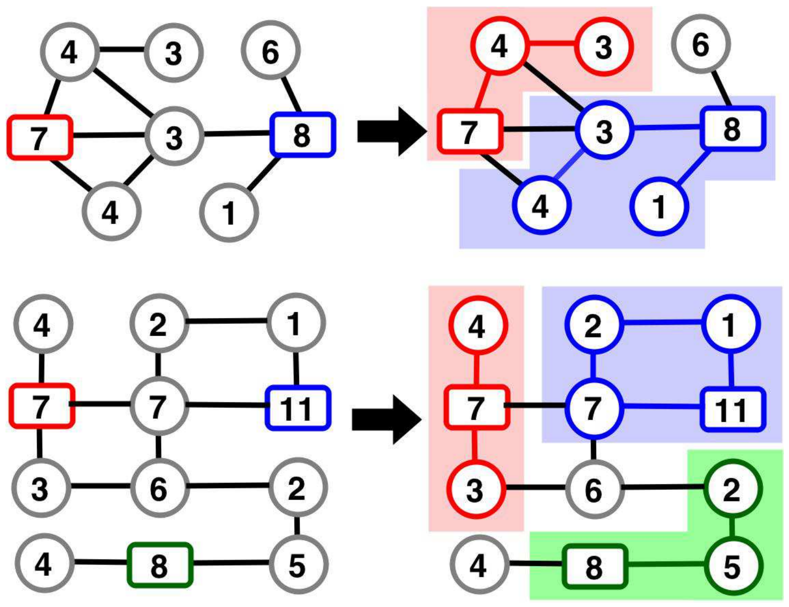


Figure 1: Examples of problem instances for the MPGSD. On the left the square nodes represent supply nodes and circles demand nodes. Numbers within the nodes correspond to supply and demand values, respectively. The right side shows the solutions, where the same color (or connected shaded set) of nodes indicates they are a part of the same partitioning.

the MPGSD is NP-hard even in the case of a graph containing only one supply node and having a star structure [16].

3. Related work

As previously stated the problem of MPGSD has been widely researched with a focus on applications in electrical distribution systems. Due to the complexity of the problem significant effort has been dedicated to finding approximate and exact solutions for specific types of graphs. It has been shown, by Ito et al. [30], that the decision problem regarding the existence of a partitioning in which all the demands are satisfied in a supply/demand graph can be solved in linear time in the case of trees. In the same paper a pseudo-polynomial-time algorithm is presented for solving MPGSD in case of trees with integer supply/demand values. The computational time of this algorithm was $O(Fn^2)$ where $F =$

$min\{\sum_{v \in V_d} dem(v), \sum_{v \in V_s} sup(v)\}$. This approach is further extended to a fully polynomial-time approximation scheme with computational cost of $O(n^5)$. A similar method has been developed, by the same authors, for series-parallel graphs with the possibility of its extension to k-trees [16]. A mixed integer program for the MPGSD has been presented in the article [31]. In this work it has been shown that such an approach, in combination with some preprocessing, is very efficient in case of tree graphs and suitable for general sparse graphs.

The research group of Nishizeki has introduced several variations of the problem and explored algorithms for finding solutions. In the parametric version [19, 32] an additional parameter $0 < r < 1$ is introduced. The goal is to create a partitioning in which all the demands are satisfied, when the demand values are scaled to $r \cdot sup(u)$. Both polynomial and pseudo-polynomial time algorithms have been presented for solving this variation in case of trees. A different direction of expanding the original problem was the addition of capacities to the edges of the graph. The capacity is introduced to limit the flow that can pass through an edge. For this problem, in case of trees, the authors have also presented algorithms of polynomial and psedo-polynomial complexity [18]. The concept of flows, for this type of problem, has been further extended to the problem of spanning distribution in trees and forests of graphs [33, 34]. Another interesting variation of the MPGSD has been presented in the article [35], in which the constraint of having only one supply node in each of the subgraphs has been substituted with a maximal allowed supply in each of them. In this version of the problem multiple supply nodes are allowed in each subgraph.

An important property of MPGSD is that there is no polynomial-time approximation scheme (PTAS) for general graphs unless $P = NP$ [16]. Only limited research has been conducted in developing algorithms for MPGSD in case of general graphs. In the work of Popa [9], an approximate algorithm for the problem of interest is given with a guarantee of a $2k$-approximation for general graphs. In this paper no asymptotic computational cost is given but it is expected that it is relatively high due to the need of calculating, and maintaining, all the paths in the graph. In our previous work a two stage greedy heuristic method for MPGSD has been developed which has an approximate computational time of $|V_d| \cdot (|V_s| + |AvgNumNeighbors(S_i)| + AvgNumConnections(u))$ [29]. In the same paper a wide range of heuristic functions has been explored and a correction procedure for improving the acquired solution has been presented. The computational experiments have shown that this approach can find approximate solutions having an error of only a few percent compared to known optimal solutions. This has been done by using different combinations of heuristic functions, for the two stages of the algorithm, and applying the correction procedure to generate multiple solutions that efficiently explore the solution space. The computational efficiency of this multi-heuristic approach was demonstrated by solving problem instances having up to 10 000 nodes within a few minutes.

To the best of our knowledge, until know only deterministic algorithms have been explored for the MPGSD. In this work we attempt to fill this gap by introducing a stochastic method for the problem of interest; to be more precise we develop an ACO algorithm. The goal of such research is to create a method that can find high quality approximate solutions within a reasonable computational time even for large scale problem instances. The second issue with current research is the lack of benchmark data and results of computational experiments. This is addressed by providing an extensive set of problem instances with known optimal solutions for trees, 3-connected graphs, series-parallel graphs and general graphs. We also provide experimental results of applying the proposed ACO method for this set of data that can be used for comparison.

4. Outline of the Greedy Algorithm

In this section we give a short overview of the greedy algorithm, for which details can be found in [28], that is used as a base for the ACO method for the MPGSD. As previously stated, the solution of the problem of interest is a set of $|\Pi| = n$ subgraphs where $n = |V_s|$ is the number of supply nodes. In the initial step of the algorithm we start with n disjunct subgraphs S_i, that only contain one supply node $S = \{s_i\}$. At each of the following steps (iterations) of the algorithm one vertex $v \in V_d$ is selected and used to expand a selected subgraph S_i. The selection of both v and S_i is performed in a way that the newly generated subgraph satisfies the constraints of being connected, disjunct and fulfills Eq. 2.

Let us define NV as the set of adjacent vertices to v in G using the following equation

$$NV(v) = \{u \mid u \in V \wedge (u, v) \in E\} \tag{5}$$

The idea is to gradually expand each of the subgraphs S_i by adding new vertices v to them. The set of potential candidates for expansion of subgraph S_i, or in other words vertices that are adjacent to S_i, can be defined using the extension of NV to subgraphs. It is important to note that as the subgraph S_i will be changed in subsequent iterations, the notation S_i^k will be used to specify the state of subgraph S_i at iteration k. Now we can extend the definition of NV with the following equation.

$$\hat{N_i^k} = NV(S_i^k) = \{u \mid u \in V \wedge \exists (v \in S_i^k)(u, v) \in E\} \tag{6}$$

It is evident that if the expansion of subgraph S_i^k is done by $v \in \hat{N_i^k}$ the newly created subgraph S_i^{k+1} will be connected, but it is not necessary that the other constraints will be satisfied. More precisely, the new S_i^{k+1} need not satisfy Eq. 2, or there may exist such an S_j^{k+1} that $S_j^{k+1} \cap S_i^{k+1} = v$ for the new subgraph. This can be avoided if instead of using set $\hat{N_i^k}$, we use a restricted set of vertices N_i^k that guarantees that the constraints will be satisfied when S_i^k is expanded using $v \in N_i^k$. We shall first define sup_i^k as the available supply for subgraph S_i at iteration k in the following equation.

$$sup_i^k = sup(v_i) - \sum_{v \in S_i^k \cap V_d} dem(v) \tag{7}$$

As previously defined we have that in Eq. 7 the node v_i is the supply node of subgraph S_i. Using sup_i^k given in Eq. 7, N_i^k is defined in the following way.

$$N_i^k = \{u \mid u \in \hat{N_i^k} \wedge dem(u) \leq sup_i^k\} \setminus \bigcup_{j=1}^{n} S_j^k \tag{8}$$

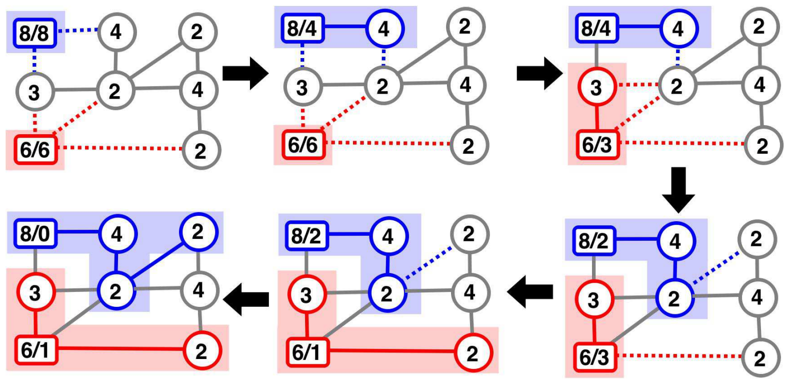


Figure 2: An example of the steps in the proposed greedy algorithm for MPGSD. The heuristic function used for subgraph selection is $h_s = sup_i^k$ and $h_n = dem(u)$ in case of node selection. Square nodes represent supply nodes and circles demand nodes. Numbers within the nodes correspond to supply and demand values, respectively. In case of supply nodes the second number represents the available supply in the subgraph. The same color (or connected shaded set) of nodes indicates they are a part of the same partition. The dashed lines are used to indicate the connections to the corrected set of neighbors N_i^k of a partition.

The sets N_i^k are used to specify the greedy algorithm for the MPGSD in combination with two heuristic functions. More precisely, at each iteration one heuristic h_s is used to select the subgraph S_i^k most suitable for expansion, and the second heuristic h_v will be used to select the best $v \in N_i^k$ to be added to S_i^k. An extensive analysis of potential heuristics is given in our previous work given in articles [28, 29]. This procedure will be repeated until it is not possible to expand any of the subgraphs. An illustration of the proposed greedy method is given in Figure 2. In this example the heuristic function for subgraph selection is $h_s = sup_i^k$ which corresponds to the subgraph with maximal available supply. The heuristic function h_n is equal to $dem(u)$, or in other words the demand node with the highest demand value.

5. Greedy Randomized Adaptive Search Procedure (GRASP)

One of the standard methods for improving the performance of a greedy algorithm is its extension to the GRASP [36] metaheuristic. The basic idea of this approach is to generate multiple solutions using a randomized greedy algorithm and further improving them by applying a local search procedure. In the proposed application of the GRASP metaheuristic to the MPGSD, the previously presented greedy algorithm is used as a basis. The randomization of the greedy algorithm has been done for both subgraph and node selection. To be more specific, the probability of selecting an element for expanding a partial solution, is proportional to its rank among the top n candidates. Formally the probability for selecting some node/subgraph i has been calculated using the following formula.

$$prob(i) = \frac{rank_h(i)}{\sum_{j=1..n} j} \tag{9}$$

In Eq. 9 $rank_h(i)$ represents the rank of candidate i based on heuristic function h.

The GRASP implementation corresponds to the pseudo-code in Algorithm 1.

Algorithm 1 GRASP

```
while (Not Stopping Criteria) do
  Initialize All $S_i$ with supply nodes
  $\Pi = \{S_1, S_2, .., S_n\}$
  repeat
    while ($Sum(|N_i|) > 0$) do
      Randomly Select $S_i$ based on rank using $h_s(S_i)$
      Randomly Select $u \in N_i$ based on rank using $h_n(u, S_i)$
      Add $u$ to $S_i$
      Update auxiliary structures
    end while
    Apply local search to $\Pi$
    Check if $\Pi$ is the best found solution
  until (NoChange)
end while
```

As it can be seen from the pseudocode, the first loop is used to generate multiple solutions until the stopping criterion is reached. In our implementation the stopping criterion is a maximal number of generated generated solutions. The inner loop corresponds to a randomized version of the previously presented greedy method. The two selections that have been performed using a heuristic function in the greedy algorithm, are now done using a probabilistic method.

Each such solution is further improved by applying the local search. The local search is the same as the one previously developed for the MPGSD [29]. The final step is simply a comparison if the newly generated solution is the best found.

6. Application of Ant Colony Optimization

In this section we present an ACO approach for solving the MPGSD, based on the greedy algorithm from Section 4. The general idea of ACO algorithms is to perform an "intelligent" randomization of an appropriate greedy algorithm for the problem of interest. There are several variations of ACO, out of which the Ant Colony System [37] is most commonly used. In it the "intelligence" comes from experience gained by previously generated solutions, which is stored in a pheromone matrix. In practice, a colony of n artificial ants generates solutions using a probabilistic algorithm based on a heuristic function and the pheromone matrix. As in the case of the greedy algorithm, an ant generates a solution by expanding a partial one through several steps. The difference is that instead of using a heuristic function it uses a probabilistic transition rule to decide what is to be added to the partial solution. The pheromone matrix stores the experience gathered by the artificial ants. This is done by applying a global and local update rule to the pheromone matrix. The global update rule is used after all n ants in the colony have generated a solution and it reinforces the selection of elements inside of the best found solution or in some variations of good solutions. The local update is performed after an ant has applied the transition rule, and its purpose is to diversify the search of the solution space by avoiding the selection of the same elements of the solution by all of the ants.

Before defining the ACO algorithm for the MPGSD, we will first state some observations regarding the greedy algorithm and the form of the solution of the problem. A solution Π of the MPGSD can also be observed as a set of pairs (s, v), which states that node v is inside subgraph S_s. In this notation we will include $(-1, v)$ for the case where v is not a member of any subgraph S_s. From this type of notation we realize that in the algorithm given in the previous section only the second stage, the selection of node v, directly specifies the elements of the solution. The purpose of the heuristic in the first stage is to make it possible to perform a good expansion of the partial solution, which is of significant importance when only one solution is generated using a deterministic algorithm. In case of an ACO algorithm this becomes less important since many solutions are generated and the "steering" in the direction of good solutions is, to a large extent, done by the pheromone matrix.

Because of this, in the proposed ACO algorithm the heuristic function at this stage will be substituted with a random selection from the set of subgraphs that can be expanded. In this way the ACO mechanism will only be dedicated to the selection of expansion nodes.

6.1. Algorithm Specification

To specify the ACO for the MPGSD we need to define the transition rule, global and local update rules. In all of the following equations we will assume that we have a randomly selected subgraph S_s with index s. We will first define the transition rule, based on the same heuristic function as in [28], defined in the following equation.

$$\eta_v = h_n(v) = dem(v) \tag{10}$$

The heuristic function η_v given in Eq. 10 states that vertices with high demand are considered more desirable. The logic behind this is that it gets harder to satisfy high demands as the algorithm progresses since the available supply decreases as new vertices are added to the subgraphs. Because of this it is better to resolve high demands early.

Using η_v we can define the transition rule for individual ants. This selection is done using a combination of deterministic and probabilistic steps. First we need to include the constraint that only vertices from the set N_i^k can be selected. We specify this constraint using the following equation.

$$p_v^k = \begin{cases} 0 & , v \notin N_s^k \\ prob_i^k & , v \in N_s^k \end{cases} \tag{11}$$

In Eq. 11, p_v^k gives us the probability of selecting node v at step k. As previously stated we only consider $v \in N_s^k$ where s is the selected subgraph, as a consequence the probability of selecting $v \notin N_s^k$ is 0. For the nodes that are elements of N_s^k their selection is done using the following formula.

$$prob_v^k = \begin{cases} 1 & , q < q_0 \ \& \ v = \arg\max\limits_{i \in N_s^k} \tau_{is}\eta_i \\ 0 & , q < q_0 \ \& \ v \neq \arg\max\limits_{i \in N_s^k} \tau_{is}\eta_i \\ \frac{\tau_{vs}\eta_v}{\sum_{i \in N_s^k} \tau_{is}\eta_i} & , q \geq q_0 \end{cases} \tag{12}$$

In Eq. 12 $prob_v^k$ gives us the probability of selecting node v at step k. The values of the pheromone matrix τ_{is} correspond to elements of the solution in the form of a vertex-subgraph pair (i, s). In the same equation parameter q_0 is used to define the exploitation/ exploration rate. Connected to it, $q \in (0, 1)$ is a random variable which specifies whether the next selected node will be deterministic or non-deterministic. In the case of the former $q < q_0$, we simply select the node v with the maximal value of $\tau_{is}\eta_i$, which results in a probability 1. If the selection is non-deterministic ($q > q_0$), the probability distribution for node selection is given in the last row of Eq. 12.

The next component of the ACO method that needs to be specified is the global update rule. The proposed ACO algorithm is based on the ant colony system, in which only the best found solution deposits pheromone after each iteration of the colony. This update is formally defined using the following equations

$$\Delta\tau = Val(\Pi') \tag{13}$$

$$\tau_i = (1-p)\tau_{vs} + p\Delta\tau \qquad ,\forall(v,s)\in\Pi' \tag{14}$$

In Eq. 13 Π' is used to note the currently best found solution. $\Delta\tau$ is used to specify the quality of the solution Π' using function Val for which we will give details in the implementation subsection. In Eq. 14, the parameter $p\in(0,1)$ is used to specify the influence of the global update rule. It is important to point out that Eq. 14 only effects the values of pheromone τ_{vs} for $(v,s)\in\Pi'$.

As previously mentioned the local update rule is applied after individual ants perform the transition rule. In our implementation the local update rule is applied after an ant i has generated a solution Π_i using the following formula

$$\tau_{vs} = \varphi\tau_{vs} \qquad ,\forall(v,s)\in\Pi_i \tag{15}$$

In Eq. 15 $\varphi\in(0,1)$ is used to specify the influence of the local update rule.

6.2. Implementation

In this section we give details of the implementation of the proposed ACO algorithm. The first necessary step is to define a suitable quality function Val for the generated solutions. This is done by using the following equations.

$$T = \sum_{v\in V_s} sup(v) \tag{16}$$

$$Val(\Pi) = \frac{1}{T - D(\Pi) + 1} \tag{17}$$

Eq. 17 states that the quality of the solution will be inversely proportional to the difference of T, the total initially available supply in G and the satisfied demand $D(\Pi)$ of partitioning Π. To avoid division by zero, one is added to this value. Using this measure, the initial value of all the pheromone matrix elements t_{vs} is set to the value $Val(\Pi_g)$, where Π_g is the solution acquired using the previously outlined greedy algorithm. More precisely, it corresponds to the method presented in [28], where the node selection heuristic is the maximal demand and the subgraph selection heuristic is the maximal available supply.

Algorithm 2 Ant colony optimization

```
Generate solution $\Pi_g$ using the greedy algorithm
Initialize the pheromone matrix $\tau$ with $Val(\Pi_g)$
while (Maximal number of iterations not reached) do
  for all $n$ ants do
    Initialize $\Pi = \{S_1, .., S_n\}$ , $S_i = \{s_i\}$
    while $\Pi$ can be expanded do
      Randomly select $S$, where $|NV(S)| > 0$
      Select $v$ for $S$ using transition rule
      $\Pi = \Pi \cup (v, S)$
      Update auxiliary structures
    end while
    Apply local update rule for $\Pi$
  end for
  Apply correction procedure to $\Pi$
  Apply global update rule for $\Pi_{best}$
end while
```

With the goal of having a better presentation of the proposed method, it is presented in the pseudo-code given in Algrothm 2.

As illustrated in the pseudo-code, the first step is generating a solution using a greedy algorithm and initializing the pheromone matrix τ. The main loop performs one iteration for the colony of ants by generating a solution for each of the n artificial ants. For each of the ants we start with the initial partitioning Π. At each iteration of the following loop, we randomly select a subgraph S, and using the transition rule a node v is selected for expansion. After each such step it is necessary to update the auxiliary structures, presented in [28], that are used to make the proposed algorithm computationally efficient.

After an ant has generated a solution Π we apply the correction procedure, presented in [29], which corresponds to a local search to improve its quality. This is done due to the fact that, in general, ACO algorithms have a problem with narrowing on local minima. It has been shown that the performance of such methods can be significantly improved if they are combined with a local search. For the newly acquired solution we apply the local update rule given in Eq. 15. After all of the ants in the colony have generated their solutions we apply the global update rule given in Eq. 14 for the best solution Π_{best} found by the algorithm for all the previous iterations.

7. Results

In this section we present the results of the computational experiments used to evaluate the performance of the proposed ACO methods. We have observed the behavior of the proposed ACO algorithm, with (ACO-C) and without (ACO) the use of a correction procedure. The comparison has been done with the previously presented deterministic greedy algorithm (Gr) and its randomized version (Gr-R). The goal of these test was to examine the effect of ACO learning mechanism. Further, both ACO methods have been compared with the GRASP metaheuristic. All the algorithms have been implemented in C# using Microsoft Visual Studio 2012. The source code and the executive files have been made available at [38]. The calculations have been done on a machine with Intel(R) Core(TM) i7-2630 QM CPU 2.00 Ghz, 4GB of DDR3-1333 RAM, running on Microsoft Windows 7 Home Premium 64-bit.

To have an extensive evaluation of the proposed algorithm tests have been conducted on a wide range of graphs. We have used 24 different graph sizes having 2-100 supply nodes and 6-2000 demand nodes. For each of the test sizes 40 different problem instances have been generated. With the goal of observing the potential dependence between the method's performance and the graph structure, we have performed tests on trees, 3-connected graphs, series-parallel graphs and general graphs. A part of these test instances have been used in the article [29] where specifics of the method for their generation are presented. It is important to note that the optimal solutions are known for each of the test instances due to the algorithm used for their generation. We have compared the average solution quality and the number of found optimal solutions for each size. All the used problem instances can be downloaded from [38].

For each of the 40 problem instances, inside of one graph size, a single run of the ACO algorithm has been performed for both versions of the method. In each of the runs the colony had 10 ants and 150 iterations have been performed. In practice this means that 1500 solutions have been generated for each test instance. The parameters for specifying the influence of the global and the local update rules had the following values $p = 0.1$ and $\varphi = 0.9$. We have used the value $q_0 = 0.9$ to define the exploitation/exploration rate. The chosen parameters correspond to the commonly used values for the ACO algorithm. In the later part of this section a detailed sensitivity analysis for these parameters is presented. In case of Gr-R and GRASP algorithms the same number of solutions (1500) have been generated. In both cases the randomization has been done using a probability distribution based on the heuristic functions used for node/subgraph selection, as presented in Section 5.

The results of the conducted computational experiments are presented in two groups of tables. In the Tables 1, 3, 5, 7 we compare ACO to GR, and GR-R for general graphs, tree graphs, 3-connected graphs and series-parallel graphs, respectively. In the second group, see Tables 2, 4, 6, 8, we compare the two ACO methods to the GRASP metaheuristic for the same types of graphs.

The values in these tables represent the average normalized error of the found solutions compared to the known optimal ones, for each of the used methods. More precisely, for each of the 40 test instances, for each graph size, the normalized error is calculated by $(Optimal - found)/Optimal \cdot 100$, and we show the average values in Tables 1, 2, 3, 4, 5, 6, 7, 8. To have a better comprehension of the performance we have also included the standard deviation and maximal errors. The last value included in these tables is the number of found optimal solutions (hits) for each graph size out of the 40 test instances.

For all types of the tested graphs, there is a very significant improvement in the quality of found solutions by the simple inclusion of randomization to Gr. The advantage of Gr-R is most notable in case of the smallest problem instances, where it frequently manages to find optimal solutions. In case of small graphs having 2 or 5 supply nodes it manages to slightly out perform ACO. The reason for this is that the ACO method tends to become trapped in some local optima and repeatedly generate the same solutions. As discussed in [25], this is a common problem with ACO implementations which can relatively easily be avoided by using multiple restarts or some kind of pheromone correction procedure. In case of the greedy algorithms the average error varies from less than 1% to 16%, and 0 to 10% in case of Gr and Gr-R, respectively. It is important to point out that the benefit of the randomization in the greedy algorithm decreases with the size of the graphs. Overall, the ACO algorithm significantly outperforms Gr-R having the average error in the range between 0-4%. The advantage of the learning mechanism inside ACO is most notable in case of the largest graphs.

The results presented in Tables 2, 4, 6, 8 show that the GRASP metaheuristic has a very good performance and significantly outperforms Gr-R. The average error ranges from 0- 8%. This approach has proven to be extremely effective in case of small graphs (2 or 5 supply nodes) where it manages to find optimal solutions in almost all the tested cases. The conducted computational experiments show that GRASP has a better performance than ACO in all but the largest problem instances (50, 100 supply nodes). In case of such large graphs the improvement of GRASP compared to Gr-R is notably smaller than in case of small graphs. Overall the ACO-C algorithm has a notably better performance when compared to the other tested methods. The conducted computational experiments show that the inclusion of a local search is essential for the performance of ACO in case of MPGSD.

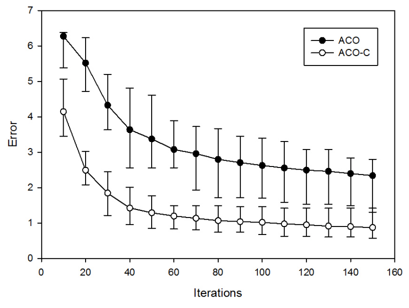
It is important to note that there is a difference in performance, when the average error is observed, of the methods for different types of graphs. The deterministic greedy algorithm has the best performance in case of general graphs and the worst for series-parallel graphs. A similar tendency has been shown for Gr-R and GRASP, especially in case of large graphs, but with higher quality of found solutions. For the other types of instances (tree, 3-connected and general graphs) the increased level of connectivity in a graph improves the performance of the Gr, Gr-R and GRASP methods. In case of the proposed ACO algorithms we have an opposite situation. For trees, when the performance was the best, the ACO method had only twice an average error higher than 1% and never higher than 2.03%. ACO-C produces even better results with never having an error greater than 1%, and having an error of less than 0.1% in 19 out of 24 graph sizes.

The results in Tables 1, 3 , 5, 7 show that the Gr and Gr-R only manage to find optimal solutions in a few cases for the smallest graphs. On the other hand the ACO-C manages to find the optimal solution for about 50% of the test instances, while ACO is close to 30%. As in the case of average errors both ACO methods have a significantly better performance for trees than other graphs. In case of trees ACO-C has found the optimal solution for 65% of the test instances, but it would rarely find ones for the largest graphs. It is interesting to note that the negative effect of the specific structure of series-parallel graphs on the performance of the Gr-R algorithm is to a large extent removed by the inclusion of the ACO mechanism. For this type of graphs the ACO-C managed to find the second highest number of optimal solutions.

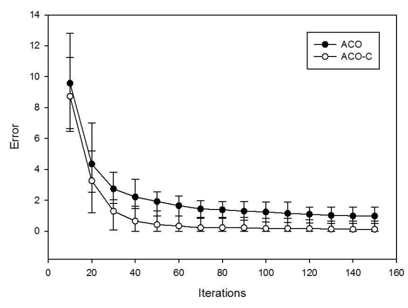
In Table 9 we give the execution times of the computational experiments given in Tables 2, 4, 6, 8. To be exact, the total computational time for solving all 40 problem instances of one graph size are presented for ACO and ACO-C. Our tests have shown that the ACO mechanization (transition, local and global update rule) was only a minor part of the computational cost compared to the greedy algorithm. Due to this fact we did not include the computational time for the greedy method in Table 9, since it is approximately equal to the one of ACO divided by 1500. The execution times of ACO, has shown a dependence on the type of the graph. The method had the shortest execution times for trees and the longest in case of general graphs. In general it would increase with the connectivity of a graph. The relative increase in calculation time of ACO-C compared to ACO had a similar behavior, in regards to connectivity, and it had the range from 1.1 to 3 times.

Due to the stochastic nature of the ACO algorithm, we have also performed multiple runs of ACO and ACO-C, with different seeds for the random number generator, on a single problem instance. In case of this type of analysis, for small problem instances both methods have a very good performance and manage to find the best solution for the vast majority of runs. The interesting cases are connected to the – harder to solve – large graph sizes. The behavior of both algorithms is similar for all tested instances in one problem size, because of which we believe it is sufficient to give a graphic illustration only for a single test instance for the different graph types in Figure 3. For all the tested graph types the ACO-C has a significantly higher speed of convergence. We can also see that both methods have a notable dependence on the selected seed of the random number generator since the difference between minimal and maximal error is 2% and 1% for ACO and ACO-C, respectively.

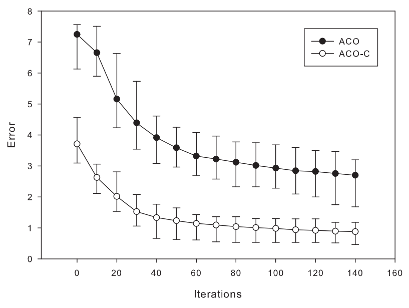
In our final group of tests we have analyzed the parameter sensitivity of the proposed ACO method. Due to the fact that there was no significant difference in the behavior of the ACO algorithm for different types of graphs the illustration is only given for general ones. For each of the parameters, tests have been conducted on graphs having different ratios between the number of supply and



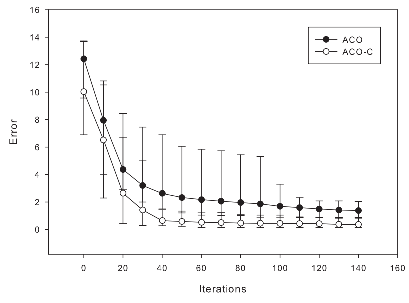
(a) General graph with 50 supply and 250 demand nodes



(b) Tree graph with 50 supply and 1000 demand nodes



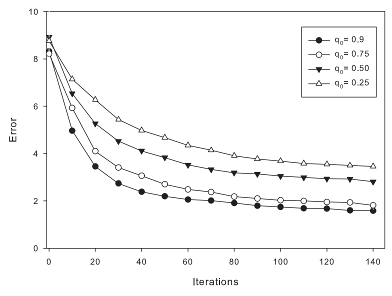
(c) 3-connected graph with 50 supply and 250 demand nodes



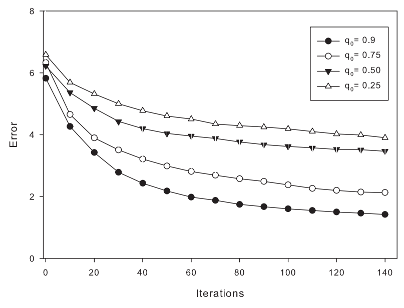
(d) Series/Parallel graph with 50 supply and 250 demand nodes

Figure 3: Illustration of the performance of ACO and ACO-C for 20 runs on different types of graphs. Each of the subgraphs shows the average error for the 20 runs at each iteration on a single problem instance. The range represents the maximal and minimal error at each step.

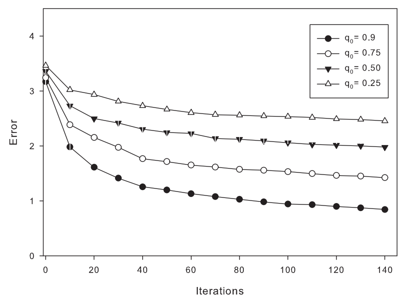
demand nodes. The results can be seen in Figures 4, 5, 6 for the parameters q_0, p, φ. As previously stated the best performance of the algorithm has been achieved for $p = 0.1$, $\varphi = 0.9$ and $q_0 = 0.9$. Figure 4 shows that the proposed algorithm is highly sensitive to changes in the exploration/exploitation rate specified by q_0. The algorithm was significantly less sensitive to changes in the values of the other two parameters. Changes to the parameter p had the least effect on the performance of the proposed method. Although the best results have been achieved for $p = 0.1$, small changes in p hardly influenced the performance of the algorithm. Even in case of $p = 0.75$ the algorithm managed



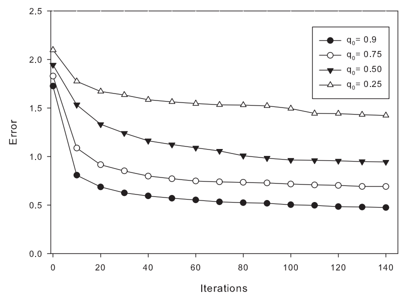
(a) 25 supply and 75 demand nodes



(b) 25 supply and 125 demand nodes



(c) 25 supply and 250 demand nodes

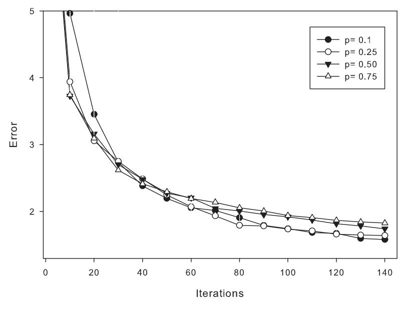


(d) 25 supply and 500 demand nodes

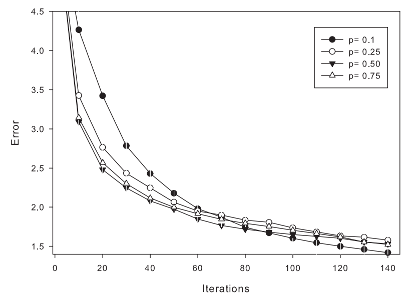
Figure 4: Sensitivity of the ACO algorithm to changes in the value of the parameter q_0 in case of general graphs for different graph sizes. The results show the average error of a single run on each of the 40 problem instances inside of one graph size at each iteration. The values of the other ACO parameters where $p = 0.1$ and $\varphi = 0.9$

to find only slightly worse results than in the best case.

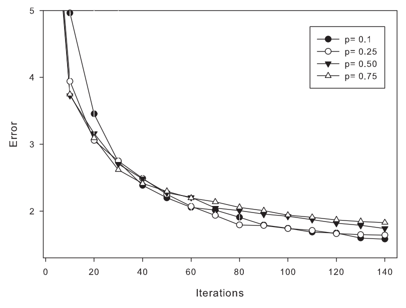
From the performed tests we can conclude that the proposed ACO algorithm is a very effective method for solving the MPGSD. The tests have also shown that, as for many others, in the case of the problem of interest the performance of the ACO algorithm can be significantly improved by adding a local search method. Finally, the quality of the solutions acquired by the ACO and ACO-C is to a certain extent dependent on the seed of the random number generator. Because of this fact, when applying the proposed method it is advisable to perform multiple runs to get the highest quality of found solutions.



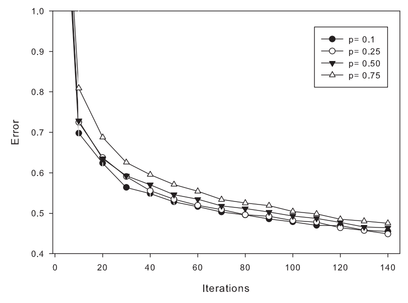
(a) 50 supply and 125 demand nodes



(b) 50 supply and 250 demand nodes



(c) 50 supply and 500 demand nodes

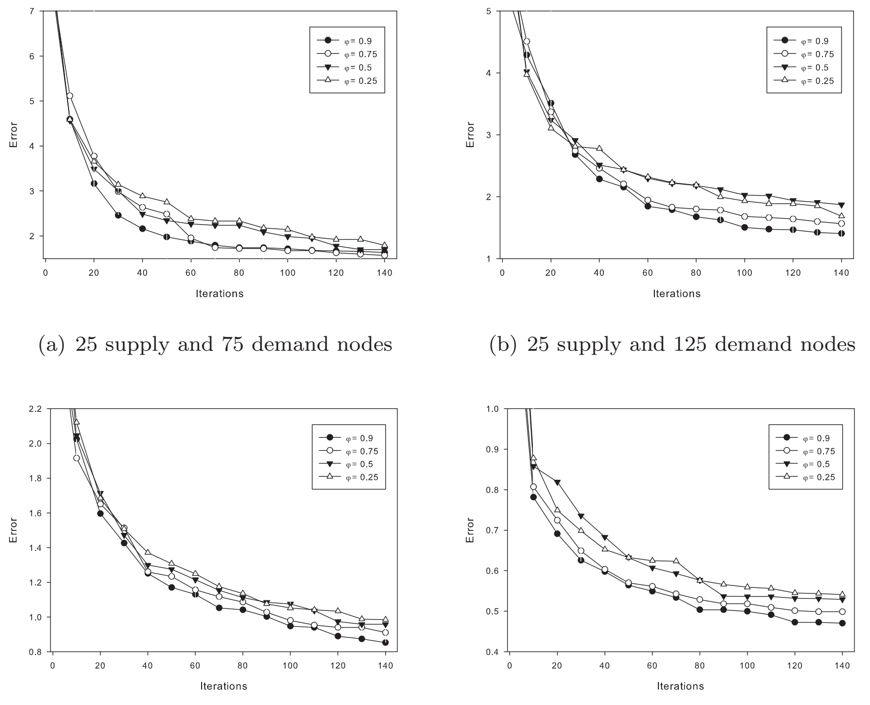


(d) 50 supply and 1000 demand nodes

Figure 5: Sensitivity of the ACO algorithm to changes in the value of the parameter p in case of general graphs for different graph sizes. The results show the average error of a single run on each of the 40 problem instances inside of one graph size at each iteration. The values of the other ACO parameters where $q_0 = 0.9$ and $\varphi = 0.9$

8. Conclusion

In this paper we have presented an ant colony optimization algorithm for solving the problem of the maximum partitioning of graphs with supply and demand. To the best of our knowledge, this is the first time that the ACO metaheuristic has been applied to this type of problem. The basic ACO algorithm has been combined with a local search to enhance the performance of the method. Our computational experiments have shown that the proposed approach managed to find the optimal solutions in more than 50% of the test problem instances, and had an average relative error of less then 0.5%. The



(a) 25 supply and 75 demand nodes

(b) 25 supply and 125 demand nodes

(c) 25 supply and 250 demand nodes

(d) 25 supply and 500 demand nodes

Figure 6: Sensitivity of the ACO algorithm to changes in the value of the parameter φ in case of general graphs for different graph sizes. The results show the average error of a single run on each of the 40 problem instances inside of one graph size at each iteration.The values of the other ACO parameters where $p = 0.1$ and $q_0 = 0.9$

tests have been performed on trees, 3-connected graphs, series-parallel graphs and general graphs and have shown that the method is most suitable for trees.

In the future we plan to adapt the method to a less constrained and a stochastic version of the problem. This type of research can prove to be very beneficial for problems appearing in the field of electrical distribution systems especially for the optimization of self-adequacy of interconnected microgrids and other related problems.

[1] N. Hatziargyriou, H. Asano, R. Iravani, C. Marnay, Microgrids, IEEE Power and Energy Magazine 5 (4) (2007) 78–94.

[2] S. Arefifar, Y. Mohamed, T. H. M. EL-Fouly, Supply-adequacy-based optimal construction of microgrids in smart distribution systems, IEEE Transactions on Smart Grid 3 (3) (2012) 1491–1502.

[3] S. Arefifar, Y.-R. Mohamed, T. EL-Fouly, Optimum microgrid design for enhancing reliability and supply-security, IEEE Transactions on Smart Grid 4 (3) (2013) 1567–1575.

[4] S. Arefifar, Y.-R. Mohamed, T. EL-Fouly, Comprehensive operational planning framework for self-healing control actions in smart distribution grids, IEEE Transactions on Power Systems 28 (4) (2013) 4192–4200.

[5] N. Boulaxis, M. Papadopoulos, Optimal feeder routing in distribution system planning using dynamic programming technique and GIS facilities, IEEE Transactions on Power Delivery 17 (1) (2002) 242–247.

[6] T. Ito, X. Zhou, T. Nishizeki, Partitioning trees of supply and demand, International Journal of Foundations of Computer Science 16 (4) (2005) 803–827.

[7] A. B. Morton, I. M. Mareels, An efficient brute-force solution to the network reconfiguration problem, IEEE Transactions on Power Delivery 15 (3) (2000) 996–1000.

[8] J.-H. Teng, C.-N. Lu, Feeder-switch relocation for customer interruption cost minimization, IEEE Transactions on Power Delivery 17 (1) (2002) 254–259.

[9] A. Popa, Modelling the power supply network - hardness and approximation, in: T.-H. Chan, L. Lau, L. Trevisan (Eds.), Theory and Applications of Models of Computation, Vol. 7876 of Lecture Notes in Computer Science, Springer, Berlin, 2013, pp. 62–71.

[10] K. Sun, D.-Z. Zheng, Q. Lu, A simulation study of OBDD-based proper splitting strategies for power systems under consideration of transient stability, IEEE Transactions on Power Systems 20 (1) (2005) 389–399.

[11] J. Li, C.-C. Liu, K. P. Schneider, Controlled partitioning of a power network considering real and reactive power balance, IEEE Transactions on Smart Grid 1 (3) (2010) 261–269.

[12] K. Andreev, H. Räcke, Balanced graph partitioning, in: Proceedings of the Sixteenth Annual ACM Symposium on Parallelism in Algorithms and Architectures, SPAA '04, ACM, New York, 2004, pp. 120–124.

[13] G. Reinelt, D. O. Theis, K. M. Wenger, Computing finest mincut partitions of a graph and application to routing problems, Discrete Applied Mathematics 156 (3) (2008) 385 – 396.

[14] E. Barnes, A. Vannelli, J. Walker, A new heuristic for partitioning the nodes of a graph, SIAM Journal on Discrete Mathematics 1 (3) (1988) 299–305.

[15] G. Reinelt, K. M. Wenger, Generating partitions of a graph into a fixed number of minimum weight cuts, Discrete Optimization 7 (12) (2010) 1 – 12.

[16] T. Ito, E. D. Demaine, X. Zhou, T. Nishizeki, Approximability of partitioning graphs with supply and demand, Journal of Discrete Algorithms 6 (4) (2008) 627 – 650.

[17] N. S. Narayanaswamy, G. Ramakrishna, Linear time algorithm for tree t-spanner in outerplanar graphs via supply-demand partition in trees, in: CoRR, 2012, abs/1210.7919.

[18] M. Kawabata, T. Nishizeki, Partitioning trees with supply, demand and edge-capacity, IEICE Transactions 96-A (6) (2013) 1036–1043.

[19] S. Morishita, T. Nishizeki, Parametric power supply networks, in: D.-Z. Du, G. Zhang (Eds.), Computing and Combinatorics, Vol. 7936 of Lecture Notes in Computer Science, Springer, Berlin, 2013, pp. 245–256.

[20] T. Ito, T. Hara, X. Zhou, T. Nishizeki, Minimum cost partitions of trees with supply and demand, Algorithmica 64 (3) (2012) 400–415.

[21] M. Dorigo, C. Blum, Ant colony optimization theory: A survey, Theoretical Computer Science 344 (2) (2005) 243–278.

[22] K. Tashkova, P. Korosec, J. Silc, A distributed multilevel ant-colony algorithm for the multi-way graph partitioning, International Journal of Bio-Inspired Computation 3 (5) (2011) 286–296.

[23] F. Comellas, E. Sapena, A multiagent algorithm for graph partitioning, in: F. Rothlauf, J. Branke, S. Cagnoni, E. Costa, C. Cotta, R. Drechsler, E. Lutton, P. Machado, J. Moore, J. Romero, G. Smith, G. Squillero, H. Takagi (Eds.), Applications of Evolutionary Computing, Vol. 3907 of Lecture Notes in Computer Science, Springer, Berlin, 2006, pp. 279–285.

[24] M. Hinne, E. Marchiori, Cutting graphs using competing ant colonies and an edge clustering heuristic, in: P. Merz, J.-K. Hao (Eds.), Evolutionary Computation in Combinatorial Optimization, Vol. 6622 of Lecture Notes in Computer Science, Springer, Berlin, 2011, pp. 60–71.

[25] R. Jovanovic, M. Tuba, An ant colony optimization algorithm with improved pheromone correction strategy for the minimum weight vertex cover problem, Applied Soft Computing 11 (8) (2011) 5360 – 5366.

[26] R. Jovanovic, M. Tuba, Ant colony optimization algorithm with pheromone correction strategy for the minimum connected dominating set problem., Computer Science and Information Systems (2013) 133–149.

[27] P. Korosec, J. Silc, B. Robic, Mesh-partitioning with the multiple ant-colony algorithm, in: M. Dorigo, M. Birattari, C. Blum, L. Gambardella, F. Mondada, T. Stützle (Eds.), Ant Colony Optimization and Swarm Intelligence, Vol. 3172 of Lecture Notes in Computer Science, Springer, Berlin, 2004, pp. 430–431.

[28] R. Jovanovic, A. Bousselham, A greedy method for optimizing the self-adequacy of microgrids presented as partitioning of graphs with supply and demand, in: The 2nd International Renewable and Sustainable Energy Conference Ouarzazate, Morocco October 17-19, 2014, IEEE conference, 2014, pp. 154–159.

[29] R. Jovanovic, A. Bousselham, S. Voss, A heuristic method for solving the problem of partitioning graphs with supply and demand, Annals of Operations Research 235 (1) (2015) 371–393.

[30] T. Ito, X. Zhou, T. Nishizeki, Partitioning trees of supply and demand, in: P. Bose, P. Morin (Eds.), Algorithms and Computation, Vol. 2518 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2002, pp. 612–623.

[31] R. Jovanovic, S. Voss, A mixed integer program for partitioning graphs with supply and demand emphasizing sparse graphs, Optimization Letters, doi:10.1007/s11590-015-0972-6.

[32] S. Morishita, T. Nishizeki, Parametric power supply networks, Journal of Combinatorial Optimization 29 (1) (2015) 1–15.

[33] K. Inoue, T. Nishizeki, Spanning distribution forests of graphs, in: Frontiers in Algorithmics, Springer, 2014, pp. 117–127.

[34] M. Kawabata, T. Nishizeki, Spanning distribution trees of graphs, IEICE Transactions on Information and Systems 97 (3) (2014) 406–412.

[35] R. Jovanovic, A. Bousselham, S. Voss, Partitioning of supply/demand graphs with capacity limitations: an ant colony approach, Journal of Combinatorial Optimization, doi:10.1007/s10878-015-9945-z.

[36] T. Feo, M. Resende, Greedy randomized adaptive search procedures, Journal of Global Optimization 6 (2) (1995) 109–133.

[37] M. Dorigo, L. M. Gambardella, Ant colony system: a cooperative learning approach to the traveling salesman problem, IEEE Transactions on Evolutionary Computation 1 (1) (1997) 53–66.

[38] R. Jovanovic, Benchmark data sets for the problem of partitioning graphs with supply and demand (2013).
URL `http://mail.ipb.ac.rs/~rakaj/home/graphsd.htm`

Table 1: Comparison of the basic (Gr) and randomized (Gr-R) greedy algorithms with the ant colony optimization (ACO) method for general graphs. In case of Gr-R and ACO 1500 solutions have been generated. The best results for each graph size are underlined.

Sup X Dem		*Avg*(*Stdev*)			*Max*			*Hits*	
	Gr	Gr-R	ACO	Gr	Gr-R	ACO	Gr	Gr-R	ACO
2 X 6	7.45(8.71)	$\underline{0.00}$(0.00)	0.28(1.77)	46.10	$\underline{0.00}$	11.36	$\underline{40}$	17	39
2 X 10	5.62(4.39)	$\underline{0.00}$(0.00)	0.24(0.60)	23.08	$\underline{0.00}$	3.04	$\underline{40}$	4	32
2 X 20	1.85(1.11)	$\underline{0.01}$(0.03)	0.09(0.13)	4.21	$\underline{0.22}$	0.46	$\underline{39}$	1	26
2 X 40	0.77(0.50)	$\underline{0.00}$(0.00)	$\underline{0.00}$(0.00)	1.74	$\underline{0.00}$	$\underline{0.00}$	$\underline{40}$	3	$\underline{40}$
5 X 15	10.88(7.77)	$\underline{0.17}$(0.51)	0.59(1.31)	38.78	$\underline{2.66}$	7.14	$\underline{34}$	0	28
5 X 25	7.89(5.99)	0.91(0.56)	$\underline{0.78}$(0.77)	34.62	$\underline{2.14}$	3.84	0	0	$\underline{7}$
5 X 50	3.89(2.62)	0.44(0.21)	$\underline{0.15}$(0.13)	10.27	0.91	$\underline{0.65}$	1	0	$\underline{8}$
5 X 100	2.01(2.54)	0.14(0.07)	$\underline{0.02}$(0.03)	13.63	0.40	$\underline{0.13}$	0	0	$\underline{26}$
10 X 30	11.53(4.44)	1.74(1.55)	$\underline{0.51}$(0.82)	23.88	6.35	$\underline{4.29}$	1	0	$\underline{19}$
10 X 50	7.36(2.79)	2.46(0.63)	$\underline{1.08}$(0.45)	14.19	3.94	$\underline{2.20}$	0	0	0
10 X 100	3.92(2.44)	1.13(0.29)	$\underline{0.28}$(0.14)	13.14	1.80	$\underline{0.69}$	0	0	0
10 X 200	2.52(2.81)	0.39(0.09)	$\underline{0.10}$(0.05)	12.98	0.60	$\underline{0.22}$	0	0	$\underline{1}$
25 X 75	12.14(3.16)	5.72(1.34)	$\underline{1.63}$(1.08)	19.23	9.61	$\underline{5.38}$	0	0	$\underline{1}$
25 X 125	8.64(2.07)	5.05(0.83)	$\underline{1.76}$(0.58)	13.64	6.64	$\underline{3.16}$	0	0	0
25 X 250	4.60(1.49)	2.36(0.42)	$\underline{0.83}$(0.19)	8.68	3.62	$\underline{1.22}$	0	0	0
25 X 500	2.81(1.37)	1.03(0.46)	$\underline{0.44}$(0.07)	6.10	3.12	$\underline{0.56}$	0	0	0
50 X 150	12.04(1.86)	8.29(0.84)	$\underline{2.20}$(0.78)	15.63	9.79	$\underline{3.91}$	0	0	0
50 X 250	8.76(1.34)	6.68(0.92)	$\underline{2.67}$(0.47)	10.80	8.72	$\underline{3.59}$	0	0	0
50 X 500	4.65(1.28)	3.10(0.45)	$\underline{1.56}$(0.23)	7.39	4.31	$\underline{2.27}$	0	0	0
50 X 1000	3.07(0.99)	1.73(0.51)	$\underline{0.73}$(0.11)	5.97	3.79	$\underline{0.99}$	0	0	0
100 X 300	11.75(1.45)	10.42(0.88)	$\underline{3.69}$(0.69)	14.61	11.89	$\underline{6.05}$	0	0	0
100 X 500	8.77(1.07)	8.06(0.63)	$\underline{3.93}$(0.60)	11.65	9.57	$\underline{6.21}$	0	0	0
100 X 1000	4.67(0.89)	4.33(0.48)	$\underline{2.29}$(0.22)	7.04	5.19	$\underline{2.71}$	0	0	0
100 X 2000	3.04(0.73)	2.60(0.63)	$\underline{1.11}$(0.22)	4.58	4.68	$\underline{1.82}$	0	0	0

Table 2: Comparison of the performance of the GRASP, ACO and ACO-C methods for general graphs. The results are presented for the case when each method has generated 1500 solutions. The best results for each graph size are underlined.

Sup X Dem		*Avg*(*Stdev*)			*Max*			*Hits*	
	GRASP	ACO	ACO-C	GRASP	ACO	ACO-C	GRASP	ACO	ACO-C
2 X 6	0.00(0.00)	0.28(1.77)	0.00(0.00)	0.00	11.36	0.00	40	39	40
2 X 10	0.00(0.00)	0.24(0.60)	0.00(0.00)	0.00	3.04	0.00	40	32	40
2 X 20	0.00(0.00)	0.09(0.13)	0.00(0.00)	0.00	0.46	0.00	40	26	40
2 X 40	0.00(0.00)	0.00(0.00)	0.00(0.00)	0.00	0.00	0.00	40	40	40
5 X 15	0.00(0.00)	0.59(1.31)	0.13(0.44)	0.00	7.14	2.22	40	28	36
5 X 25	0.04(0.12)	0.78(0.77)	0.22(0.29)	0.55	3.84	1.07	35	7	21
5 X 50	0.00(0.00)	0.15(0.13)	0.01(0.03)	0.10	0.65	0.10	38	8	35
5 X 100	0.00(0.00)	0.02(0.03)	0.00(0.00)	0.00	0.13	0.00	40	26	40
10 X 30	0.03(0.14)	0.51(0.82)	0.16(0.40)	0.85	4.29	1.60	37	19	32
10 X 50	0.53(0.33)	1.08(0.45)	0.26(0.26)	1.19	2.20	0.90	4	0	13
10 X 100	0.14(0.06)	0.28(0.14)	0.05(0.05)	0.31	0.69	0.18	3	0	18
10 X 200	0.01(0.01)	0.10(0.05)	0.00(0.00)	0.02	0.22	0.00	31	1	40
25 X 75	1.08(0.57)	1.63(1.08)	0.28(0.29)	2.32	5.38	1.14	0	1	12
25 X 125	1.45(0.35)	1.76(0.58)	0.51(0.31)	2.50	3.16	1.49	0	0	0
25 X 250	0.48(0.10)	0.83(0.19)	0.13(0.06)	0.72	1.22	0.23	0	0	0
25 X 500	0.12(0.04)	0.44(0.07)	0.01(0.02)	0.21	0.56	0.06	0	0	11
50 X 150	2.37(0.61)	2.20(0.78)	0.46(0.40)	3.91	3.91	1.78	0	0	3
50 X 250	2.83(0.34)	2.67(0.47)	0.84(0.22)	3.16	3.59	1.42	0	0	0
50 X 500	1.79(0.21)	1.56(0.23)	0.31(0.07)	0.97	2.27	0.50	0	0	0
50 X 1000	0.93(0.19)	0.73(0.11)	0.06(0.02)	0.70	0.99	0.13	0	0	0
100 X 300	3.78(0.59)	3.69(0.69)	0.90(0.42)	5.22	6.05	2.02	0	0	0
100 X 500	4.06(0.38)	3.93(0.60)	1.42(0.28)	3.96	6.21	2.13	0	0	0
100 X 1000	3.10(0.33)	2.29(0.22)	0.60(0.07)	1.48	2.71	0.74	0	0	0
100 X 2000	1.41(0.43)	1.11(0.22)	0.14(0.04)	1.35	1.82	0.27	0	0	0

Table 3: Comparison of the basic (Gr) and randomized (Gr-R) greedy algorithms with the ant colony optimization (ACO) method for tree graphs. In case of Gr-R and ACO 1500 solutions have been generated. The best results for each graph size are underlined.

Sup X Dem	*Avg(Stdev)*			*Max*			*Hits*		
	Gr	Gr-R	ACO	Gr	Gr-R	ACO	Gr	Gr-R	ACO
2 X 6	1.67(5.92)	$\underline{0.00}$(0.00)	$\underline{0.00}$(0.00)	26.37	$\underline{0.00}$	$\underline{0.00}$	37	$\underline{40}$	$\underline{40}$
2 X 10	5.46(8.02)	$\underline{0.00}$(0.00)	0.11(0.50)	35.14	$\underline{0.00}$	3.08	16	$\underline{40}$	37
2 X 20	8.71(9.11)	$\underline{0.03}$(0.15)	0.09(0.34)	28.94	$\underline{0.78}$	2.13	3	$\underline{38}$	35
2 X 40	6.09(7.65)	$\underline{0.00}$(0.00)	0.05(0.14)	30.58	$\underline{0.00}$	0.57	3	$\underline{40}$	34
5 X 15	8.47(8.71)	$\underline{0.00}$(0.00)	0.01(0.04)	27.19	$\underline{0.00}$	0.27	13	$\underline{40}$	39
5 X 25	7.87(6.04)	$\underline{0.00}$(0.00)	0.10(0.25)	21.80	$\underline{0.00}$	1.11	1	$\underline{40}$	33
5 X 50	10.63(6.99)	0.12(0.18)	$\underline{0.07}$(0.16)	29.60	$\underline{0.45}$	0.89	0	26	$\underline{28}$
5 X 100	16.43(11.22)	0.18(0.81)	$\underline{0.12}$(0.64)	50.93	5.23	$\underline{4.09}$	0	25	$\underline{30}$
10 X 30	8.66(6.44)	$\underline{0.00}$(0.00)	0.09(0.25)	27.17	$\underline{0.00}$	1.13	2	$\underline{40}$	$\underline{34}$
10 X 50	9.67(5.59)	0.10(0.27)	$\underline{0.07}$(0.17)	29.53	1.62	0.$\underline{70}$	0	29	$\underline{31}$
10 X 100	11.40(6.33)	0.44(0.62)	$\underline{0.09}$(0.13)	26.73	3.30	$\underline{0.53}$	0	9	$\underline{19}$
10 X 200	13.92(6.58)	0.79(1.08)	$\underline{0.27}$(1.15)	26.02	7.44	$\underline{6.71}$	0	1	$\underline{23}$
25 X 75	9.52(4.87)	0.78(0.74)	$\underline{0.18}$(0.35)	22.49	3.14	$\underline{1.44}$	0	5	$\underline{26}$
25 X 125	10.79(3.83)	2.20(1.38	$\underline{0.15}$(0.16)	17.29	5.92	$\underline{0.63}$	0	0	$\underline{12}$
25 X 250	10.68(3.22)	2.94(1.34)	$\underline{0.29}$(0.60)	20.23	6.47	$\underline{2.73}$	0	0	$\underline{9}$
25 X 500	11.64(3.93)	3.65(1.48)	$\underline{0.48}$(0.69)	19.37	6.98	$\underline{2.72}$	0	0	$\underline{2}$
50 X 150	8.66(2.93)	3.09(1.14)	$\underline{0.15}$(0.18)	17.04	5.33	$\underline{0.76}$	0	0	$\underline{13}$
50 X 250	10.20(3.07)	4.69(1.25)	$\underline{0.31}$(0.29)	18.72	8.28	$\underline{1.23}$	0	0	$\underline{2}$
50 X 500	11.92(3.03)	5.72(1.23)	$\underline{0.44}$(0.50)	18.84	7.97	$\underline{2.21}$	0	0	0
50 X 1000	12.75(2.41)	6.45(1.43)	$\underline{1.09}$(0.85)	18.30	9.61	$\underline{3.61}$	0	0	0
100 X 300	9.83(1.99)	5.57(0.84)	$\underline{0.27}$(0.18)	14.11	8.19	$\underline{0.81}$	0	0	$\underline{2}$
100 X 500	10.26(1.79)	6.68(1.01)	$\underline{0.56}$(0.35)	14.43	8.61	$\underline{1.62}$	0	0	0
100 X 1000	11.18(1.82)	7.91(1.19)	$\underline{1.05}$(0.51)	14.55	10.59	$\underline{2.25}$	0	0	0
100 X 2000	12.07(1.86)	7.98(1.23)	$\underline{2.03}$(0.75)	17.49	10.47	$\underline{3.69}$	0	0	0

Table 4: Comparison of the performance of the GRASP, ACO and ACO-C methods for tree graphs. The results are presented for the case when each method has generated 1500 solutions. The best results for each graph size are underlined.

Sup X Dem	*Avg*(*Stdev*)			*Max*			*Hits*		
	GRASP	ACO	ACO-C	GRASP	ACO	ACO-C	GRASP	ACO	ACO-C
2 X 6	$\underline{0.00}$(0.00)	$\underline{0.00}$(0.00)	$\underline{0.00}$(0.00)	$\underline{0.00}$	$\underline{0.00}$	$\underline{0.00}$	$\underline{40}$	$\underline{40}$	$\underline{40}$
2 X 10	$\underline{0.00}$(0.00)	0.11(0.50)	0.02(0.13)	$\underline{0.00}$	3.08	0.85	$\underline{40}$	37	39
2 X 20	0.02(0.10)	0.09(0.34)	$\underline{0.01}$(0.07)	0.62	2.13	0.43	38	35	$\underline{39}$
2 X 40	$\underline{0.00}$(0.00)	0.05(0.14)	$\underline{0.00}$(0.00)	$\underline{0.00}$	0.57	$\underline{0.00}$	$\underline{40}$	34	$\underline{40}$
5 X 15	$\underline{0.00}$(0.00)	0.01(0.04)	$\underline{0.00}$(0.00)	$\underline{0.00}$	0.27	$\underline{0.00}$	$\underline{40}$	39	$\underline{40}$
5 X 25	$\underline{0.00}$(0.00)	0.10(0.25)	0.07(0.29)	$\underline{0.00}$	1.11	1.49	$\underline{40}$	33	37
5 X 50	$\underline{0.00}$(0.00)	0.07(0.16)	0.04(0.15)	$\underline{0.00}$	0.89	0.89	$\underline{40}$	28	35
5 X 100	0.00(0.01)	0.12(0.64)	$\underline{0.00}$(0.00)	0.05	4.09	$\underline{0.00}$	39	30	$\underline{40}$
10 X 30	$\underline{0.00}$(0.00)	0.09(0.25)	0.01(0.06)	$\underline{0.00}$	1.13	0.37	$\underline{40}$	34	39
10 X 50	$\underline{0.01}$(0.04)	0.07(0.17)	0.07(0.21)	$\underline{0.26}$	0.70	1.08	$\underline{38}$	31	34
10 X 100	$\underline{0.00}$(0.00)	0.09(0.13)	0.03(0.09)	$\underline{0.05}$	0.53	0.48	$\underline{38}$	19	33
10 X 200	0.26(0.63)	0.27(1.15)	$\underline{0.25}$(1.15)	6.71	6.71	6.71	33	23	$\underline{37}$
25 X 75	$\underline{0.00}$(0.00)	0.18(0.35)	0.03(0.12)	$\underline{0.00}$	1.44	0.73	$\underline{40}$	26	36
25 X 125	0.08(0.16)	0.15(0.16)	$\underline{0.06}$(0.13)	0.56	0.63	$\underline{0.47}$	21	12	$\underline{27}$
25 X 250	0.24(0.47)	0.29(0.60)	$\underline{0.06}$(0.23)	2.68	2.73	$\underline{1.31}$	11	9	$\underline{30}$
25 X 500	1.87(1.58)	0.48(0.69)	$\underline{0.14}$(0.34)	5.21	2.72	$\underline{1.27}$	0	2	$\underline{30}$
50 X 150	0.06(0.06)	0.15(0.18)	$\underline{0.04}$(0.09)	0.53	0.76	$\underline{0.46}$	29	13	$\underline{30}$
50 X 250	0.31(0.35)	0.31(0.29)	$\underline{0.07}$(0.09)	1.76	1.23	$\underline{0.39}$	6	2	$\underline{17}$
50 X 500	2.41(1.32)	0.44(0.50)	$\underline{0.05}$(0.13)	5.08	2.21	$\underline{0.79}$	0	0	$\underline{11}$
50 X 1000	4.65(1.25)	1.09(0.85)	$\underline{0.51}$(0.60)	7.70	3.61	$\underline{1.92}$	0	0	$\underline{10}$
100 X 300	0.29(0.21)	0.27(0.18)	$\underline{0.09}$(0.15)	0.80	0.81	$\underline{0.64}$	5	2	$\underline{17}$
100 X 500	2.01(0.93)	0.56(0.35)	$\underline{0.08}$(0.06)	4.22	1.62	$\underline{0.21}$	0	0	$\underline{3}$
100 X 1000	4.94(1.42)	1.05(0.51)	$\underline{0.18}$(0.29)	7.60	2.25	$\underline{1.55}$	0	0	$\underline{3}$
100 X 2000	6.58(1.13)	2.03(0.75)	$\underline{0.97}$(0.75)	9.30	3.69	$\underline{3.99}$	0	0	0

Table 5: Comparison of the basic (Gr) and randomized (Gr-R) greedy algorithms with the ant colony optimization (ACO) method for 3-connected graphs. In case of Gr-R and ACO 1500 solutions have been generated. The best results for each graph size are underlined.

Sup X Dem	*Avg*(*Stdev*)			*Max*			*Hits*		
	Gr	Gr-R	ACO	Gr	Gr-R	ACO	Gr	Gr-R	ACO
2 X 6	7.03(10.06)	$\underline{0.00}$(0.00)	$\underline{0.00}$(0.00)	44.07	$\underline{0.00}$	$\underline{0.00}$	22	$\underline{40}$	$\underline{40}$
2 X 10	6.16(5.59)	$\underline{0.00}$(0.00)	0.21(0.53)	29.76	$\underline{0.00}$	2.40	3	$\underline{40}$	33
2 X 20	2.65(1.56)	$\underline{0.01}$(0.04)	0.15(0.21)	9.05	$\underline{0.23}$	0.73	0	$\underline{39}$	22
2 X 40	1.10(2.50)	$\underline{0.00}$(0.00)	$\underline{0.00}$(0.00)	16.35	$\underline{0.00}$	$\underline{0.00}$	2	$\underline{40}$	$\underline{40}$
5 X 15	9.06(6.78)	$\underline{0.09}$(0.38)	0.11(0.32)	30.39	2.27	$\underline{1.59}$	4	$\underline{37}$	35
5 X 25	8.10(3.76)	0.51(0.48)	$\underline{0.47}$(0.44)	16.51	2.05	$\underline{1.48}$	1	10	$\underline{12}$
5 X 50	4.85(3.88)	0.48(0.22)	$\underline{0.29}$(0.20)	13.98	0.91	$\underline{0.79}$	0	1	$\underline{5}$
5 X 100	1.15(1.19)	0.13(0.06)	$\underline{0.03}$(0.05)	6.86	0.27	$\underline{0.22}$	0	1	$\underline{25}$
10 X 30	11.60(5.93)	1.12(1.14)	$\underline{0.35}$(0.68)	25.42	3.86	$\underline{2.83}$	0	11	$\underline{25}$
10 X 50	8.97(3.77)	2.48(0.87)	$\underline{1.14}$(0.84)	18.29	4.61	$\underline{4.23}$	0	0	0
10 X 100	4.47(2.35)	1.25(0.23)	$\underline{0.34}$(0.19)	9.66	1.74	$\underline{0.90}$	0	0	0
10 X 200	1.73(1.88)	0.39(0.07)	$\underline{0.09}$(0.04)	7.70	0.56	$\underline{0.21}$	0	0	$\underline{1}$
25 X 75	10.71(2.55)	5.58(1.29	$\underline{0.87}$(0.62)	16.28	9.74	$\underline{2.33}$	0	0	$\underline{6}$
25 X 125	8.85(1.85)	4.74(0.81	$\underline{1.52}$(0.48)	13.34	6.20	$\underline{2.51}$	0	0	0
25 X 250	4.62(1.45)	2.37(0.30)	$\underline{0.80}$(0.20)	8.77	2.96	$\underline{1.35}$	0	0	0
25 X 500	1.83(1.10)	0.73(0.06	$\underline{0.42}$(0.07)	5.24	0.85	$\underline{0.56}$	0	0	0
50 X 150	11.98(1.90)	8.05(1.08)	$\underline{1.58}$(0.62)	16.85	10.55	$\underline{3.23}$	0	0	0
50 X 250	9.66(1.60)	6.77(0.79)	$\underline{2.36}$(0.53)	14.13	8.42	$\underline{4.08}$	0	0	0
50 X 500	4.62(1.04)	3.28(0.40)	$\underline{1.52}$(0.21)	6.81	4.44	$\underline{1.87}$	0	0	0
50 X 1000	1.67(0.68)	0.99(0.10)	$\underline{0.69}$(0.07)	3.24	1.30	$\underline{0.87}$	0	0	0
100 X 300	11.87(1.37)	9.93(0.79	$\underline{2.70}$(0.66)	15.00	11.51	$\underline{4.00}$	0	0	0
100 X 500	9.18(1.34)	8.31(0.62)	$\underline{3.73}$(0.47)	13.35	9.35	$\underline{4.94}$	0	0	0
100 X 1000	4.82(0.76)	4.33(0.39)	$\underline{2.25}$(0.21)	6.38	5.14	$\underline{2.78}$	0	0	0
100 X 2000	1.87(0.50)	1.39(0.15)	$\underline{0.94}$(0.07)	3.69	1.76	$\underline{1.06}$	0	0	0

Table 6: Comparison of the performance of the GRASP, ACO and ACO-C methods for 3-connected graphs. The results are presented for the case when each method has generated 1500 solutions. The best results for each graph size are underlined.

Sup X Dem	*Avg*(*Stdev*)			*Max*			*Hits*		
	GRASP	ACO	ACO-C	GRASP	ACO	ACO-C	GRASP	ACO	ACO-C
2 X 6	0.00(0.00)	0.00(0.00)	0.00(0.00)	0.00	0.00	0.00	40	40	40
2 X 10	0.00(0.00)	0.21(0.53)	0.12(0.66)	0.00	2.40	4.23	40	33	38
2 X 20	0.00(0.00)	0.15(0.21)	0.02(0.07)	0.00	0.73	0.24	40	22	36
2 X 40	0.00(0.00)	0.00(0.00)	0.00(0.00)	0.00	0.00	0.00	40	40	40
5 X 15	0.00(0.00)	0.11(0.32)	0.01(0.05)	0.00	1.59	0.32	40	35	39
5 X 25	0.07(0.19)	0.47(0.44)	0.26(0.40)	0.82	1.48	1.43	34	12	22
5 X 50	0.03(0.05)	0.29(0.20)	0.07(0.08)	0.19	0.79	0.27	27	5	20
5 X 100	0.00(0.00)	0.03(0.05)	0.00(0.00)	0.00	0.22	0.00	40	25	40
10 X 30	0.04(0.27)	0.35(0.68)	0.10(0.34)	1.73	2.83	1.73	39	25	35
10 X 50	0.41(0.27)	1.14(0.84)	0.27(0.29)	1.99	4.23	1.23	3	0	12
10 X 100	0.18(0.08)	0.34(0.19)	0.09(0.08)	0.40	0.90	0.31	1	0	6
10 X 200	0.01(0.01)	0.09(0.04)	0.00(0.01)	0.05	0.21	0.04	37	1	38
25 X 75	0.61(0.42)	0.87(0.62)	0.21(0.31)	1.64	2.33	1.06	5	6	23
25 X 125	1.31(0.36)	1.52(0.48)	0.47(0.22)	2.36	2.51	1.01	0	0	0
25 X 250	0.49(0.08)	0.80(0.20)	0.12(0.06)	0.64	1.35	0.30	0	0	0
25 X 500	0.07(0.02)	0.42(0.07)	0.01(0.01)	0.10	0.56	0.05	0	0	17
50 X 150	1.59(0.59)	1.58(0.62)	0.19(0.22)	3.73	3.23	0.72	0	0	10
50 X 250	2.18(0.34	2.36(0.53)	0.77(0.29)	3.73	4.08	1.55	0	0	0
50 X 500	1.78(0.08)	1.52(0.21)	0.33(0.09)	0.94	1.87	0.54	0	0	0
50 X 1000	0.83(0.02)	0.69(0.07)	0.05(0.01)	0.17	0.87	0.08	0	0	0
100 X 300	3.01(0.45)	2.70(0.66)	0.44(0.29)	4.14	4.00	1.16	0	0	2
100 X 500	3.92(0.60)	3.73(0.47)	1.21(0.30)	3.51	4.94	1.90	0	0	0
100 X 1000	2.41(0.26)	2.25(0.21)	0.58(0.08)	1.16	2.78	0.76	0	0	0
100 X 2000	1.18(0.12)	0.94(0.07)	0.10(0.01)	0.22	1.06	0.12	0	0	0

Table 7: Comparison of the basic (Gr) and randomized (Gr-R) greedy algorithms with the ant colony optimization (ACO) method for Series/parallel graphs. In case of Gr-R and ACO 1500 solutions have been generated. The best results for each graph size are underlined.

Sup X Dem	*Avg*(*Stdev*)			*Max*			*Hits*		
	Gr	Gr-R	ACO	Gr	Gr-R	ACO	Gr	Gr-R	ACO
2 X 6	8.94(12.18)	0.00(0.00)	0.00(0.00)	37.82	0.00	0.00	24	40	40
2 X 10	7.00(5.84)	0.00(0.00)	0.10(0.39)	25.85	0.00	1.84	7	40	37
2 X 20	6.80(7.98)	0.10(0.36)	0.14(0.28)	31.28	1.69	1.48	2	35	26
2 X 40	4.28(7.70)	0.00(0.02)	0.04(0.12)	42.28	0.11	0.68	2	39	35
5 X 15	9.77(7.61)	0.00(0.00)	0.12(0.46)	26.52	0.00	2.66	8	40	35
5 X 25	8.37(5.45)	0.31(0.57)	0.26(0.57)	24.72	2.40	2.40	1	26	27
5 X 50	8.39(5.78)	0.43(0.34)	0.24(0.30)	27.61	1.67	1.49	0	5	12
5 X 100	8.98(7.24)	0.43(0.40)	0.09(0.16)	27.79	1.85	0.91	0	2	19
10 X 30	12.30(7.09)	0.30(0.50)	0.22(0.43)	26.86	2.07	2.07	1	24	26
10 X 50	11.97(6.48)	1.46(0.89)	0.38(0.48)	35.05	4.02	2.37	0	1	9
10 X 100	9.05(4.16)	1.53(0.90)	0.42(0.47)	23.06	3.96	2.22	0	0	7
10 X 200	11.40(5.03)	1.50(0.77)	0.27(0.30)	23.70	4.15	1.67	0	0	1
25 X 75	12.82(3.72)	2.93(1.25)	0.30(0.37)	19.35	6.12	1.60	0	0	9
25 X 125	12.20(3.34)	4.64(1.53)	0.60(0.59)	20.71	8.71	2.45	0	0	1
25 X 250	11.80(3.71)	4.20(1.37)	1.01(0.81)	22.43	8.25	4.81	0	0	0
25 X 500	11.65(4.27)	4.56(2.28)	1.18(1.06)	27.93	13.67	5.15	0	0	0
50 X 150	12.05(3.24)	6.09(1.23)	0.63(0.48)	18.73	8.72	1.98	0	0	3
50 X 250	11.63(2.40)	6.85(1.25)	1.19(0.61)	18.63	9.83	3.29	0	0	0
50 X 500	12.01(2.39)	7.31(1.48)	1.59(0.73)	17.19	10.95	4.99	0	0	0
50 X 1000	11.75(2.50)	6.86(1.84)	2.11(1.14)	18.06	12.49	5.47	0	0	0
100 X 300	12.29(1.83)	9.00(1.06)	1.06(0.44)	16.48	10.99	1.94	0	0	0
100 X 500	11.64(1.92)	9.32(1.15)	1.94(0.64)	15.79	12.68	4.26	0	0	0
100 X 1000	11.84(1.83)	9.49(1.33)	2.74(0.72)	16.43	12.62	4.33	0	0	0
100 X 2000	11.52(1.81)	8.79(1.23)	3.09(0.93)	15.12	11.45	5.08	0	0	0

Table 8: Comparison of the performance of the GRASP, ACO and ACO-C methods for Series/parallel graphs. The results are presented for the case when each method has generated 1500 solutions. The best results for each graph size are underlined.

Sup X Dem	*Avg*(*Stdev*)			*Max*			*Hits*		
	GRASP	ACO	ACO-C	GRASP	ACO	ACO-C	GRASP	ACO	ACO-C
2 X 6	$\underline{0.00}$(0.00)	$\underline{0.00}$(0.00)	$\underline{0.00}$(0.00)	$\underline{0.00}$	$\underline{0.00}$	$\underline{0.00}$	$\underline{40}$	$\underline{40}$	$\underline{40}$
2 X 10	$\underline{0.00}$(0.00)	0.10(0.39)	0.08(0.35)	$\underline{0.00}$	1.84	1.84	$\underline{40}$	37	38
2 X 20	$\underline{0.03}$(0.14)	0.14(0.28)	0.08(0.25)	$\underline{0.85}$	1.48	1.48	$\underline{38}$	26	32
2 X 40	$\underline{0.00}$(0.00)	0.04(0.12)	$\underline{0.00}$(0.00)	$\underline{0.00}$	0.68	$\underline{0.00}$	$\underline{40}$	35	$\underline{40}$
5 X 15	$\underline{0.00}$(0.00)	0.12(0.46)	0.02(0.10)	$\underline{0.00}$	2.66	0.61	$\underline{40}$	35	39
5 X 25	$\underline{0.00}$(0.03)	0.31(0.57)	0.09(0.20)	$\underline{0.19}$	2.40	1.11	$\underline{39}$	26	28
5 X 50	$\underline{0.01}$(0.04)	0.24(0.30)	0.04(0.07)	$\underline{0.19}$	1.49	0.35	$\underline{36}$	12	27
5 X 100	0.05(0.13)	0.09(0.16)	$\underline{0.03}$(0.11)	0.95	0.91	$\underline{0.64}$	33	19	$\underline{35}$
10 X 30	$\underline{0.03}$(0.19)	0.22(0.43)	0.04(0.21)	$\underline{1.20}$	2.07	1.31	$\underline{39}$	26	37
10 X 50	0.09(0.18)	0.38(0.48)	$\underline{0.06}$(0.12)	0.88	2.37	$\underline{0.51}$	27	9	$\underline{28}$
10 X 100	0.14(0.21)	0.42(0.47)	$\underline{0.10}$(0.17)	1.05	2.22	$\underline{0.66}$	11	7	$\underline{18}$
10 X 200	0.14(0.17)	0.27(0.30)	$\underline{0.04}$(0.09)	0.89	1.67	$\underline{0.36}$	3	1	$\underline{26}$
25 X 75	0.15(0.19)	0.30(0.37)	$\underline{0.10}$(0.22)	$\underline{0.75}$	1.60	0.81	15	9	$\underline{27}$
25 X 125	0.58(0.48)	0.60(0.59)	$\underline{0.08}$(0.11)	1.84	2.45	$\underline{0.52}$	3	1	$\underline{19}$
25 X 250	0.83(0.82)	1.01(0.81)	$\underline{0.20}$(0.43)	5.41	4.81	$\underline{2.65}$	0	0	$\underline{9}$
25 X 500	1.98(2.13)	1.18(1.06)	$\underline{0.44}$(1.18)	8.48	$\underline{5.15}$	6.24	0	0	$\underline{6}$
50 X 150	0.70(0.47)	0.63(0.48)	$\underline{0.08}$(0.15)	2.02	1.98	$\underline{0.69}$	1	3	$\underline{26}$
50 X 250	1.80(0.98)	1.19(0.61)	$\underline{0.23}$(0.19)	4.52	3.29	$\underline{0.\ 74}$	0	0	$\underline{3}$
50 X 500	4.04(1.76)	1.59(0.73)	$\underline{0.44}$(0.67)	7.94	4.99	$\underline{3.48}$	0	0	0
50 X 1000	5.01(2.12)	2.11(1.14)	$\underline{0.75}$(1.03)	11.19	5.47	$\underline{4.96}$	0	0	0
100 X 300	2.36(1.17)	1.06(0.44)	$\underline{0.12}$(0.16)	5.36	1.94	$\underline{0.84}$	0	0	$\underline{9}$
100 X 500	5.05(1.39)	1.94(0.64)	$\underline{0.36}$(0.21)	7.70	4.26	$\underline{0.88}$	0	0	0
100 X 1000	6.96(1.47)	2.74(0.72)	$\underline{0.83}$(0.49)	11.04	4.33	$\underline{2.22}$	0	0	0
100 X 2000	7.68(1.26)	3.09(0.93)	$\underline{1.44}$(0.96)	10.89	5.08	$\underline{3.71}$	0	0	0

Table 9: Comparison of execution times for ACO and ACO-C for different types of graphs. The total execution time for solving 40 test instances for one graph size is given in seconds. Both algorithms have performed 150 iterations which is equivalent to the generation of 1500 solutions

Sup X Dem	Trees		3-Connected Graphs		General Graphs		Series-parallel Graphs	
	ACO	ACO-C	ACO	ACO-C	ACO	ACO-C	ACO	ACO-C
2 X 6	0.6	0.7	0.5	0.7	0.6	0.8	0.5	0.6
2 X 10	0.7	1.0	0.8	1.2	0.8	1.2	0.7	1.0
2 X 20	1.4	1.9	1.6	2.5	2.0	2.8	1.5	2.4
2 X 40	3.1	5.7	4.1	7.6	4.9	9.1	3.8	6.5
5 X 15	1.0	1.5	1.2	2.0	1.3	2.2	1.1	2.0
5 X 25	1.7	2.5	2.0	3.5	2.3	3.7	2.0	3.3
5 X 50	3.7	6.0	4.5	8.6	5.3	10.8	4.5	8.2
5 X 100	8.4	16.7	11.1	27.9	13.5	32.4	12.1	26.3
10 X 30	2.3	3.6	2.6	4.4	2.7	4.9	2.7	4.5
10 X 50	3.8	6.0	4.5	7.8	4.7	9.0	4.5	8.4
10 X 100	7.7	13.6	9.7	21.9	11.1	27.1	10.3	23.3
10 X 200	17.6	37.1	23.7	60.2	28.0	70.2	26.1	60.3
25 X 75	6.8	12.1	7.7	15.6	7.9	17.2	7.9	16.6
25 X 125	11.4	20.3	13.1	31.8	14.2	35.1	13.8	32.8
25 X 250	24.0	47.3	29.5	80.3	33.6	93.6	32.2	73.9
25 X 500	55.4	112.1	67.2	165.5	77.8	203.0	74.0	156.9
50 X 150	17.3	36.6	19.3	49.5	19.8	46.1	19.7	51.7
50 X 250	28.4	53.6	32.3	85.5	34.2	97.3	32.8	74.1
50 X 500	58.7	119.7	69.8	202.4	79.4	224.4	74.2	159.6
50 X 1000	131.7	240.1	179.1	470.3	200.9	555.9	187.8	379.0
100 X 300	51.1	121.6	54.8	130.1	56.3	144.4	56.7	129.6
100 X 500	86.1	148.3	97.5	254.7	100.0	274.9	95.9	190.1
100 X 1000	180.9	304.2	220.6	595.0	229.6	659.7	223.7	423.0
100 X 2000	392.8	639.3	531.3	1442.2	596.0	1748.8	512.6	984.3