Noname manuscript No.
(will be inserted by the editor)

Partitioning of Supply/Demand Graphs with Capacity Limitations - An Ant Colony Approach

Raka Jovanovic · Abdelkader Bousselham · Stefan Voß

Received: date / Accepted: date

Abstract In recent years there has been a growing interest for the problem of the minimal partitioning of graphs with supply and demand, due to its close connection to electrical distribution systems, especially in the context of smartgrids. In this paper we present a new version of the problem which is more suitable for practical applications in modeling such systems. More precisely, the constraint of having a unique supply node in a subgraph (partition) is substituted with a limit on the number of subgraphs and the capacity for each of them. The problem is initially solved by a two stage greedy method. With the goal of further improving the quality of found solutions, a corresponding GRASP and an ant colony optimization algorithm are developed. Due to the novelty of the problem, we include a description of a method for generating test instances with known optimal solutions. In our computational experiments we evaluate the performance of the proposed algorithms on both trees and general graphs. The tests show that the proposed ant colony approach manages to frequently find optimal solutions. It has an average relative error of less than 2% when compared to known optimal solutions. Moreover, it outperform the GRASP.

Raka Jovanovic
Qatar Environment and Energy Research Institute, Hamad bin Khalifa University, PO Box 5825, Doha, Qatar
Tel.: +974-665-13631
E-mail: rjovanovic@qf.org.qa

Abdelkader Bousselham
Qatar Environment and Energy Research Institute, Hamad bin Khalifa University, PO Box 5825, Doha, Qatar
E-mail: abousselham@qf.org.qa

Stefan Voß Institute of Information Systems, University of Hamburg, Von-Melle-Park 5, 20146 Hamburg, Germany , and
Escuela de Ingeniera Industrial, Pontificia Universidad Católica de Valparaíso, Chile
E-mail: stefan.voss@uni-hamburg.de

Keywords Ant Colony Optimization · Microgrid · Graph Partitioning · Demand Vertex · Supply Vertex · Combinatorial Optimization · GRASP

1 Introduction

Recently, the concept of interconnected microgrids has had an increased influence on the development of smartgrids (Hatziargyriou et al, 2007). In case of such systems the electrical grid is partitioned into highly independent subsections (microgrids). The new topology has brought the necessity of solving many new optimization problems related to practical applications. Some examples are the maximizing of self-adequacy (Arefifar et al, 2012), reliability, supply-security (Arefifar et al, 2013b) and the potential for self-healing (Arefifar et al, 2013a) of such systems. The mutual independence of individual microgrids, inside of an interconnected system, has resulted in a wide range of positive characteristics. For instance, 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 independence corresponds to the level of power exchange between the connected microgrids. More precisely, if there is a low level of power exchange there is a high level of independence. This characteristic is formally defined as the maximization of self-adequacy.

Certain global properties of the electrical grids can be efficiently modeled and optimized using simplified graph models. Such models frequently correspond to different types of graph partitioning problems. Some examples are the use of a balanced partitioning (Andreev and Räcke, 2004), minimizing the number or weight of cuts (Reinelt et al, 2008; Barnes et al, 1988), or by limiting the number of cuts (Reinelt and Wenger, 2010). The problem of finding the maximal partitioning of graphs with supply and demand (MPGSD) has shown to be closely related to the maximization of self-adequacy of interconnected microgrids (Jovanovic and Bousselham, 2014; Jovanovic et al, 2015). Focus of the research for MPGSD, to a large extent, has been dedicated to the theoretical aspects of this problem (Ito et al, 2008; Narayanaswamy and Ramakrishna, 2012; Ito et al, 2005; Kawabata and Nishizeki, 2013). A method for finding solutions with a guarantee of a $2k$-approximation for general graphs has been presented in Popa (2013).

The published research for this problem has significant drawbacks that hinder its wider application in the field of microgrids. One reason is that a large part of previous research has focused on solving the MPGSD for specific types of graphs like trees (Narayanaswamy and Ramakrishna, 2012; Ito et al, 2005; Kawabata and Nishizeki, 2013) and series-parallel graphs (Ito et al, 2008). However, the main obstacle for the application of MPGSD within the field of microgrids is in the problem definition itself. More precisely, the constraint of having a unique supply for each demand node does not correlate to electrical grids with multiple power sources. The existing variations of the original problem, like the parametric version (Morishita and Nishizeki, 2013) and the

one with the inclusion of additional capacity constraints (Ito et al, 2012) are designed to address different properties of the system.

In this paper we present a new version of the MPGSD without the constraint of having a unique supply node for each demand node. Instead a limit is given for the maximal allowed supply in a partition (subgraph) and the total number of allowed partitions. For the newly defined problem a two stage greedy algorithm is developed. In the initial stage the MPGSD is solved, and in the second one suitable partitions are merged. As it is a well known fact, although greedy algorithms are generally computationally inexpensive the quality of acquired results is often relatively low. There is a wide range of approaches for enhancing the performance of such algorithms like the use of the GRASP (Feo and Resende, 1995; Hart and Shogan, 1987) or pilot (Voß et al, 2005) methods. The ant colony optimization (ACO) (Dorigo and Blum, 2005) metaheuristic has shown to be very suitable for such a task in case of graph problems. In the context of the problem of interest, ACO has formerly proven to be an effective approach for solving various graph partitioning problems like the multiway (Tashkova et al, 2011) and balanced (Comellas and Sapena, 2006) ones, and also for the closely related graph cutting (Hinne and Marchiori, 2011) and covering (Jovanovic and Tuba, 2011, 2013) problems. In our previous work, the ACO method has been successfully applied to the MPGSD (Jovanovic et al, 2014). This gave us inspiration to develop an ACO algorithm for the problem of interest. The developed ACO method, like the greedy algorithm, consists of two stages but both of them use the same pheromone matrix. To evaluate the proposed algorithm, tests have been performed on general graphs and trees, with various sizes. The computational experiments show that the proposed ACO approach frequently manages to find optimal solutions and has a small average error when compared to known optimal solutions. Moreover, we propose a GRASP to have a proper comparison with the ACO. Numerical results show that, while the ACO approach outperforms the GRASP, it can benefit from being hybridized with the local search incorporated in the GRASP.

The paper is organized as follows. In the second section we give the definition for the problem of Maximal Partitioning of Supply/Demand Graphs with Capacity Limitations (MPGSD-CL). In the following section the two stage greedy algorithm is presented. In addition, we describe a GRASP based on the proposed greedy method. In Section 4, we provide details of the corresponding ACO algorithm. In the subsequent sections we propose a data generation mechanism to provide problem instances with known optimal solutions. Moreover, we discuss results of our computational experiments and provide some conclusions.

2 Maximal Partitioning of Supply/Demand Graphs with Capacity Limitations

The MPGSD-CL 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 set of disjoint subgraphs $\Pi = \{S_1, S_2, .., S_N\}$ of the graph G for a fixed value N that satisfies the following constraints. All the subgraphs in Π must be connected subgraphs. Each subgraph S_i consists of supply and demand nodes and they must have a total supply greater or equal to its total demand. The total supply in each of the subgraphs S_i must be no more than a fixed limit M_s. 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 $S_i \in \Pi$

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

$$\sum_{v \in S_i \cap V_s} sup(v) \leq M_s \tag{3}$$

$$S_i \cap S_j = \emptyset \quad , \; i \neq j \tag{4}$$

$$S_i \;\; is \;\; connected \tag{5}$$

$$|\Pi| = N \tag{6}$$

Eq. 3 is used to define the constraint of the capacity limit. It states that the sum of all $sup(v)$ for supply nodes v in S_i is less than or equal to the limit M_s. In the problem definition only a single integer limit value M_s is used for all the partitions. An illustration of the MPGSD-LC is given in Figure 1. Although we do not include any proof for NP-hardness of MPGSD-LC, it is important to mention that such a proof has been given for MPGSD. This is true even for graphs containing only one supply node and having a star structure (Ito et al, 2008).

3 Greedy Algorithm

In this section we present the proposed two stage greedy algorithm. As previously stated the first stage consists of solving the MPGSD, and in the second one different subgraphs are merged. This merger is done in a way that the newly created subgraphs satisfy the maximal supply constraint given in Eq. 3. As it will be seen, in practice, these two stages are performed in a loop since the application of one can influence the other.

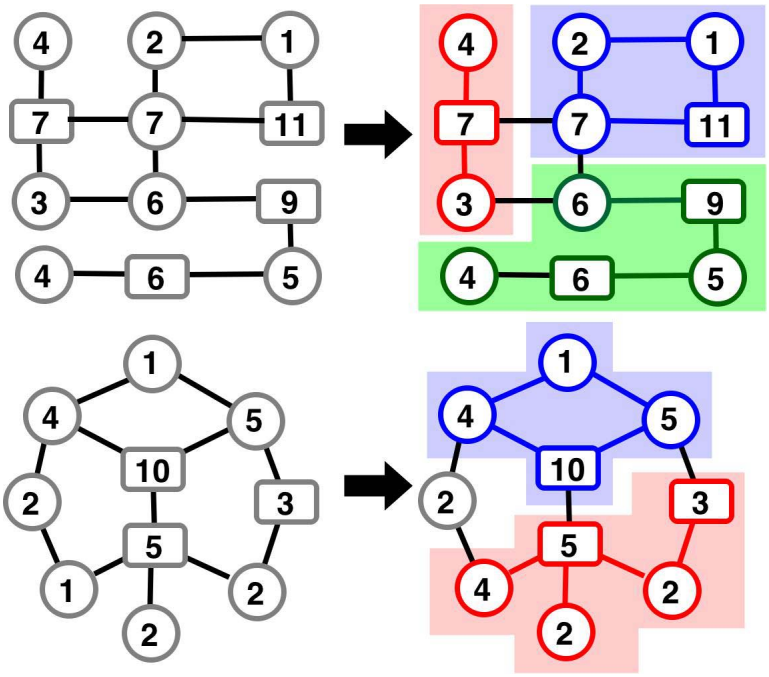

Fig. 1 Examples of problem instances for the MPGSD-LC. On the left the square nodes represent supply nodes and circles demand nodes. Numbers within the nodes correspond to supply and demand values, respectively. The right hand side shows the solutions, where the same color (or connected shaded set) of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

3.1 Outline of Greedy Algorithm for MPGSD

For the sake of completeness in this subsection we give an outline of the greedy algorithm for MPGSD, for which details can be found in (Jovanovic and Bousselham, 2014). As previously stated, the solution of the MPGSD consists of $|\Pi| = n$ subgraphs where $n = |V_s|$ is the number of supply nodes. We wish to mention once more, that for MPGSD-LC this need not be the case. At the initialization step of the algorithm we start with n seeds for the n disjunct subgraphs S_i. Each of the seeds, the starting state of a partition, is one distinct supply node $S_i = \{s_i\}$. At each of the following steps (iterations) of the algorithm a subgraph S_i is selected, and it is expanded with some vertex $v \in V_d$. The selection of both v and S_i is performed in a way that the expanded 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

$$NV(v) = \{u \mid u \in V \wedge (u, v) \in E\} \tag{7}$$

The idea of the proposed algorithm is to gradually expand the subgraphs in Π by the addition of different demand nodes v. Let us say that at some step k, where we are expanding subgraph S_i, the set of potential candidates for this task consists of vertices adjacent to S_i. This set 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. The extension of NV for subgraphs is given in the following equation.

$$\hat{N_i^k} = NV(S_i^k) = \{u \mid (u \in V \setminus S_i^k) \wedge (\exists v \in S_i^k : (u, v) \in E)\} \tag{8}$$

If the expansion of subgraph S_i^k is done using some $v \in \hat{N_i^k}$ the newly created subgraph S_i^{k+1} will be connected. But the selection of the expanding node from the set $\hat{N_i^k}$ does not guarantee the satisfaction of the other constraints. More precisely, the new subgraph S_i^{k+1} need not satisfy Eq. 2, or there may exist such an S_j^{k+1} that $S_j^{k+1} \cap S_i^{k+1} = v$. To avoid this, we use a restricted set of candidates $N_i^k \subset \hat{N_i^k}$ which insures the satisfaction of these constraints. We shall first define Sup_i^k as the available supply for subgraph S_i at iteration k in the following way.

$$Sup_i^k = \sum_{v \in S_i^k \cap V_s} sup(v) - \sum_{v \in S_i^k \cap V_d} dem(v) \tag{9}$$

Now N_i^k is defined using the following equation.

$$\bar{N_i^k} = \{u \mid u \in \hat{N_i^k} \wedge sup(u) \leq Sup_i^k\}$$
$$N_i^k = \bar{N_i^k} \setminus \bigcup_{j=1}^{n} S_j^k \tag{10}$$

The greedy algorithm for MPGSD uses two separate heuristic functions in combination with the sets N_i^k. At each step of the algorithm, first a heuristic h_s is used to select the subgraph S_i^k most suitable for expansion. Using a second heuristic h_n the best $v \in N_i^k$ is selected and added to S_i^k. An in-depth analysis of potential heuristics is given in our previous work (Jovanovic and Bousselham, 2014; Jovanovic et al, 2015). This procedure will be repeated until it is not possible to expand any of the subgraphs.

3.2 Merging procedure

The second stage of the greedy algorithm for solving the MPGSD-LC consists in merging suitable subgraphs. To be exact, we say that the merger of subgraphs S_i and S_j results in a subgraph S_m with a set of nodes $S_m = \{u \mid (u \in S_i) \vee (u \in S_j)\}$. The reason for the merger stage is that the number of subgraphs in a solution of MPGSD, $|V_s| = n$, is generally greater than the one for MPGSD-LC since $N \leq n$. The partition created by a merger should satisfy

connectivity and capacity limit constraints given in Eq. 3. Similar as in the case of the greedy algorithm for MPGSD, we need to define NS_i^k as the set of neighboring subgraphs to S_i^k. This can be done in the following way

$$TSup_i^k = \sum_{v \in S_i^k \cap V_s} sup(v) \tag{11}$$

$$NS_i^k = NS(S_i^k) = \{S_j^k \mid (\exists u \in S_j^k : u \in \hat{N}_i^k) \wedge (TSup_i^k + TSup_j^k \leq M_s)\} \tag{12}$$

In Eq. 11 $TSup_i^k$ represents the total supply in subgraph S_i^k. Eq. 12 gives us the set of subgraphs NS_i^k that can be merged with subgraph S_i^k, where the resulting subgraph will satisfy the necessary constraints. To be more precise, NS_i^k consists of neighboring subgraphs with a small enough total supply.

The merger procedure consists in consecutive merger steps. Although the merger operation is symmetric, for clarity of presentation we will say that at each step we are expanding subgraph S_i^k with subgraph S_j^k. There is a wide range of potential heuristics h_{se} and h_e for selecting the subgraph S_i^k that is to be expanded, and the subgraph S_j^k that will be merged with S_i^k. Such heuristics would follow a similar logic to the ones presented in articles by Jovanovic and Bousselham (2014) and Jovanovic et al (2015). Due to the fact that we are using these heuristics only as a basis for an ACO algorithm we will use two basic ones based on the total supply of subgraphs.

$$h_{se}(S_i^k) = TSup_i^k \tag{13}$$

$$h_e(S_i^k) = TSup_i^k \tag{14}$$

At each step of the merging procedure the subgraph with a maximal value of $h_{se}(S_i^k)$ and $NS_i^k \neq \emptyset$ is selected for expansion. The reason for this is twofold. As there is a limited number N of allowed subgraphs, we wish to avoid the early creation of several partitions that have a large value of $TSup$. The rationale for this is that such partitions have a limited potential for merger, and as a consequence there can be a large waste of supply. The second reason is that a subgraph with a high value of $TSup$, has the smallest set of potential expansion candidates. As a consequence, it is easy for them to be cut off from the rest of the graph, or in other words loose all potential candidates for merger, by expansion of other subgraphs.

The heuristic function h_e given in Eq. 14 states that subgraphs with high value of total supply are considered more desirable. The logic behind this is that it gets harder to merge subgraphs with a high value $TSup$ as the algorithm progresses since the available supply for merger decreases as more mergers are performed. Because of this it is better to resolve high total supplies earlier.

3.3 Complete greedy algorithm

As previously stated, the greedy algorithm for MPGSD-LC combines the use of the two stages presented in the preceding subsections. With the goal of

having a clearer presentation of the proposed method, the greedy algorithm is also given in Algorithm 1 in the form of a pseudo code.

Algorithm 1 Greedy algorithm

```
Initialize All S_i with supply nodes
Π = {S_1, S_2, .., S_n}

repeat
  while (Sum(|N_i|) > 0) do
    Select S_i using h_s(S_i)
    Select u ∈ N_i using h_n(u, S_i)
    Add u to S_i
    Update auxiliary structures
  end while
  while (Sum(|NS_i|) > 0) do
    Select S_i using h_se(S_i)
    Select S_j ∈ NS_i using h_e(S_j)
    S_i = S_i ∪ S_j
    Π = Π \ {S_j}
    Update auxiliary structures
  end while
until (NoChange)
Set Π to N subgraphs with maximal values of sup_i
```

From the pseudocode we can see that the algorithm starts with a set Π consisting of n subgraphs only having a single supply node s_i. Following is the main loop that first performs all the possible subgraph expansions using demand nodes. Afterwards all the possible mergers are done. The use of the loop comes from the fact that these two stages are interconnected, or in other words expansion steps can create the possibility of new mergers and vice versa. For instance, an expansion step for S_i^k can increase the number of neighboring subgraphs NS_i^k. Similarly, when two subgraphs are merged it is possible that the newly created one can be expanded.

The main loop is exited after an iteration in which no change has occurred to the partitioning. The final step is just selecting the N subgraphs that cover the most demand. It is important to point out that for the proposed algorithm to be computationally effective it is necessary to use some auxiliary structures similar to the ones used in (Jovanovic and Bousselham, 2014; Jovanovic et al, 2015). An illustration of the algorithm is given in Figure 2.

Note that the performance of the proposed greedy algorithm can be improved by having a more complex strategy for the order of expansion and merger operations. As previously stated, the greedy method has been developed as a basis for an ACO algorithm. We did not consider such details since it is expected that the use of ACO will give a significantly higher level of improvement.

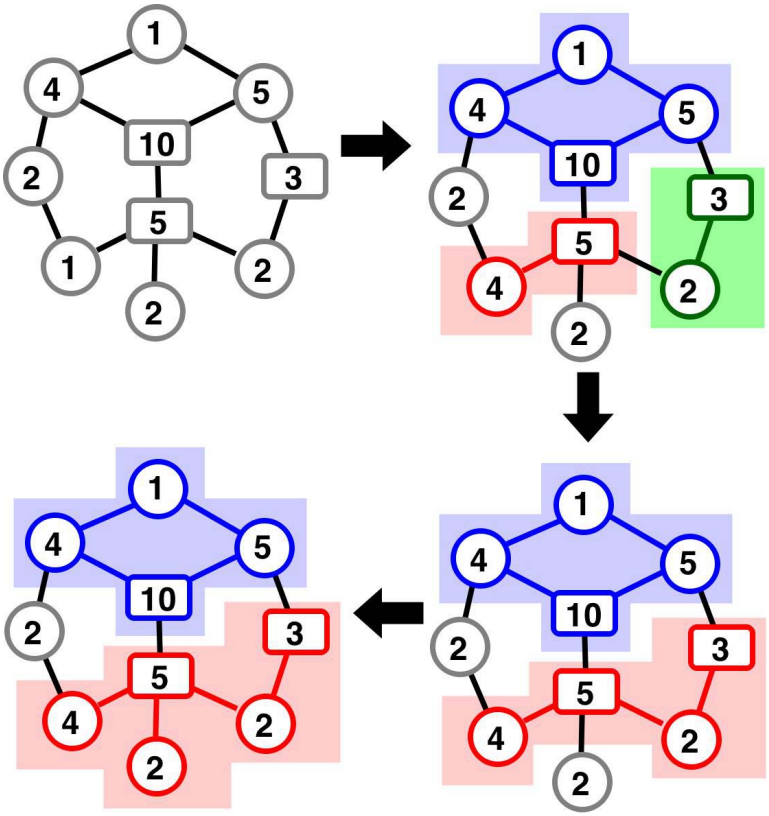

Fig. 2 Illustration of the greedy algorithm for MPGSD-LC. In all the steps squares/circles represent supply/demand vertices. Grey color is used for unassigned vertices. Other colors are used to indicate which vertices belong to the same partitions. The first step represents the problem setup, the second one shows the solution of the MPGSD. The next step shows a merger step, and the last step shows an additional expansion made possible by the merger.

3.4 Greedy Randomized Adaptive Search Procedure (GRASP)

One common method for improving the performance of a greedy algorithm is its extension to the GRASP (Feo and Resende, 1995) 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. For the problem of interest, the previously presented greedy algorithm is used as a basis for the GRASP. The randomization of the greedy algorithm has been done for both stages of the algorithm. 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. The GRASP implementation corresponds to the pseudo-code in Algorithm 2.

As it can be seen from the pseudocode, the first loop is used to generate multiple solutions until the stopping criteria is reached. The stopping criteria

Algorithm 2 GRASP

```
while (Not Stopping Criteria) do
  Initialize All S_i with supply nodes
  Π = {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 ∈ N_i based on rank using h_n(u, S_i)
      Add u to S_i
      Update auxiliary structures
    end while
    while (Sum(|NS_i|) > 0) do
      Randomly Select S_i based on rank using h_se(S_i)
      Randomly Select S_j ∈ NS_i based on rank using h_e(S_j)
      S_i = S_i ∪ S_j
      Π = Π \ {S_j}
      Update auxiliary structures
    end while
  until (NoChange)
  Set Π to N subgraphs with maximal values of sup_i
  Apply local search to Π
  Check if Π is the best found solution
end while
```

used was that a maximal number of solutions has been generated. The inner loop corresponds to a randomized version of the previously presented greedy method. In practice, four selections (two in the expansion stage and two in the merger stage) 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 (Jovanovic et al, 2015). The final step is simply a comparison if the newly generated solution is the best found.

4 Application of Ant Colony Optimization

In this section we present an ACO approach for solving the MPGSD-LC, based on the previously presented greedy algorithm. The basic concept of an ACO algorithm is to perform an "intelligent" randomization of the corresponding greedy algorithm. Due to the extensive application of ACO several variations of the algorithm have been developed, out of which the Ant Colony System (Dorigo and Gambardella, 1997) is most commonly used. The "intelligent" behavior of ACO comes from experience gained by previously generated solutions, which is stored in a pheromone matrix. ACO algorithms use a colony of n artificial ants which generate solutions. Each of the artificial ants generates a solution by expanding a partial one through some steps. Contrary to the greedy method, a probabilistic transition rule is used to decide how to expand the partial solution instead of a heuristic function.

The essential part of an ACO algorithm is the pheromone matrix, which has the purpose of storing the experience gathered by the artificial ants. To be exact, the pheromone matrix holds the information on which parts of potential solutions usually belong to high quality ones. This is done by applying the global and local update rules to the pheromone matrix. The purpose of the global update rule is to intensify the selection of elements inside of the best found solution or in some variations of good solutions. In other words, the global update rule strengthens the search in the neighborhood of the global best. This rule is applied after all n ants in the colony have generated their solutions. The goal of the local update rule is to diversify the search of the solution space. This is done by avoiding the repeated selection of the same solution elements by all of the ants. This rule is generally performed after an ant has applied the transition rule or generated a complete solution. To fully specify the ACO for the MPGSD-LC we need to define the pheromone matrix; transition, global and local update rules. Unlike the typical application of ACO, separate transition rules are used in the expansion and merger stages of the algorithm.

4.1 Pheromone Matrix

To be able to define the ACO algorithm for MPGSD-LC we first need to define what an element (part) of the solution represents. A solution Π of the MPGSD can also be observed as a set of pairs (s, v), which states that a demand node v is inside subgraph S_s with a supply node s. The same concept can be extended to the case of MPGSD-LC. The solution Π can be seen as a set of pairs (s, v), where a supply or demand node v is in a subgraph containing supply node s. The solution of a MPGSD-LC Π can formally be written as follows.

$$\Pi = \bigcup_{j=1}^{N} \left[(S_j \cap V_s) \times S_j \right] \tag{15}$$

To make this type of notation complete we include the pairs having the form $(-1, v)$ for the case where v is not a member of any subgraph.

As previously stated the solution Π of the MPGSD can be viewed as a set of pairs (s, v) where s is a supply vertex and v is a demand vertex. In the appropriate greedy algorithm the partial solution Π will be expanded with some pair (s, v) at each iteration. As a consequence the elements of the pheromone matrix, in an ACO algorithm, can naturally be defined as τ_{vs} which corresponds to the pair (s, v) (Jovanovic et al, 2014). The problem with observing the solution of MPGSD-LC in the same way is that at each step of the previously described greedy method, the partial solution will not necessarily be expanded by only one such pair but frequently with many. For instance, if a demand node d is added to a subgraph with supply nodes s_1, s_2 the partial solution will be expanded with pairs (s_1, d) and (s_2, d). In case of

the merger procedure the partial solution will be expanded with a much higher number of such elements.

From another point of view, at each step of the greedy algorithm the partial solution is expanded either with one demand node in case of an expansion; or by a subgraph in case of a merger. As a consequence, in case of an ACO algorithm, the pheromone matrix only needs to provide information on how to make such a selection. To achieve this we can use the same pheromone matrix as in the case of MPGSD, but with a specific method for retrieving values from it. This will be done differently for the expansion and merger stages.

We shall first analyze how the pheromone matrix should be used in the expansion stage. Let us assume that subgraph S_s is being expanded and we wish to retrieve the information from the pheromone matrix on how desirable node v is. In case of MPGSD this can be directly done by taking the value τ_{vs} from the pheromone matrix (Jovanovic et al, 2014). It is not possible to directly extend this approach to MPGSD-LC, since subgraph S_s may have several supply nodes, but we need to use the following formula.

$$\hat{\tau}_{vs} = \max_{v \in S_s \cap V_s} \tau_{vi} \tag{16}$$

In Eq. 16 $\hat{\tau}_{vs}$ represents the extracted pheromone value, and τ_{vi} is the value from the pheromone matrix. $\hat{\tau}_{vs}$ is equal to the maximal element of the pheromone matrix that has a demand node v and a supply node inside of S_s. We can understand that the element of pheromone matrix τ_{ij} gives us the desirability of having node i in the same subgraph as supply node j. Eq. 16 states that only the strongest "connection" between the supply nodes in S_s and v is considered. The main reason for such a choice is that a supply node j will be attracting the same group of demand nodes independently from which subgraph it belongs to.

We use the same logic for defining the procedure for extracting the pheromone value for a merger step using the following equation.

$$\overline{\tau}_{es} = \max_{\substack{v \in S_s \cap V_s \\ u \in S_e \cap V_s}} \tau_{uv} \tag{17}$$

In Eq. 17 $\overline{\tau}_{is}$ represents the extracted pheromone value for a merger between subgraphs S_e and S_s. This value will be equal to the maximal element of the pheromone matrix that has a supply inside S_e and a supply node inside of S_s.

4.2 Transition Rules

Before giving details of the transition rule for the expansion stage, we will first give a note on some observations on the structure of the solution of the MPGSD. Such observation can be extended to the case of the problem with limited capacity. If the solution is represented as a set of pairs (s, v) we can

recognize that in the corresponding greedy algorithm (Jovanovic and Bousselham, 2014; Jovanovic et al, 2015) only the second stage, the selection of node v using heuristic function h_n, fully determines the elements of the solution. To clarify, the heuristic for subgraph selection is used to make it possible to perform a good expansion of the partial solution. For a deterministic algorithm, when only one solution is generated, this is of essential importance but in case of an ACO one this is less crucial. This is due to the fact that many solutions are generated and the "steering" in the direction of good ones is, to a large extent, done by the pheromone matrix. In the view of this fact, in the proposed ACO algorithm, the heuristic function h_s is substituted with a random selection from the set of subgraphs that can be expanded. As a result, the ACO mechanism is only used for the selection of expansion nodes.

Let us assume that we have randomly selected subgraph S_s with an index s for expansion. The transition rule of an ACO algorithm is dependent on a heuristic function and the pheromone matrix. In our previous work (Jovanovic et al, 2015), an extensive analysis of potential heuristics has been presented. In the proposed ACO algorithm we will use the same one as Jovanovic and Bousselham (2014), defined in the following equation.

$$\eta_v = h_n(v) = dem(v) \tag{18}$$

The heuristic function η_v given in Eq. 18 states that vertices with high demand are considered more desirable. The reason for this is that it becomes harder to satisfy high demands as the algorithm progresses since the available supply decreases as new vertices are added to the subgraphs. Using the heuristic function given in Eq. 18 we can define the transition rule for individual ants. The selection of nodes is done from the set N_i^k to make certain that all the necessary constraints are satisfied. Formally we can write this as follows

$$p_v^k = \begin{cases} 0 & , v \notin N_s^k \\ prob_i^k & , v \in N_s^k \end{cases} \tag{19}$$

In Eq. 19, p_v^k gives the probability of selecting node v at step k. Since it is only allowed to select a node $v \in N_s^k$ where S_s is the selected subgraph, the probability of selecting $v \notin N_s^k$ is 0. The selection of a node $v \in N_s^k$ is performed based on the following formula.

$$prob_v^k = \begin{cases} 1 & , q > q_0 \ \ \& \ \ v = \arg\max\limits_{i \in N_s^k} \hat{\tau}_{is}\eta_i \\ 0 & , q > q_0 \ \ \& \ \ v \neq \arg\max\limits_{i \in N_s^k} \hat{\tau}_{is}\eta_i \\ \frac{\hat{\tau}_{vs}\eta_v}{\sum_{i \in N_s^k} \hat{\tau}_{is}\eta_i} & , q \leq q_0 \end{cases} \tag{20}$$

Eq. 20 gives the probability $prob_v^k$ of selecting node v at step k. The q_0 is the standard ACO parameter that specifies the balance between the exploitation/ exploration rate. Coupled to it, $q \in (0, 1)$ is a random variable which determines whether the selection is done in a deterministic or non-deterministic way. In the case of the former $q > q_0$, we simply select the node v with the

maximal value of $\hat{\tau}_{is}\eta_i$, 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. 20.

The transition rule for the merger stage will be defined in a similar way. First, we will consider that subgraph S_s has been selected for expansion using the heuristic function h_{se}. The chosen heuristic function used in the merger transition rule is $\overline{\eta}_e = h_e(S_e)$. Using $\overline{\eta}_s$ we can define the transition rule for individual ants. The selection of a subgraph for a merger must be done from the set NS_s^k to make certain that all the necessary constraints are satisfied. Formally we can write this as follows

$$p_e^k = \begin{cases} 0 & , S_e \notin NS_s^k \\ prob_e^k & , S_e \in NS_s^k \end{cases} \tag{21}$$

In Eq. 21, p_e^k gives the probability of selecting subgraph S_e at step k. Since it is only allowed to select a subgraph $S_e \in NS_s^k$, where S_s is the subgraph selected for expansion, the probability of selecting $S_e \notin NS_s^k$ is 0. The selection of a subgraph $S_e \in NS_s^k$ is performed based on the following formula.

$$prob_e^k = \begin{cases} 1 & , q > q_0 \ \ \& \ \ v = \arg\max\limits_{S_i \in NS_s^k} \overline{\tau}_{is}\overline{\eta}_i \\ 0 & , q > q_0 \ \ \& \ \ v \neq \arg\max\limits_{S_i \in NS_s^k} \overline{\tau}_{is}\overline{\eta}_i \\ \frac{\overline{\tau}_{vs}\overline{\eta}_v}{\sum_{S_i \in NS_s^k} \overline{\tau}_{is}\overline{\eta}_i} & , q \leq q_0 \end{cases} \tag{22}$$

Eq. 22 gives the probability $prob_e^k$ of selecting subgraph S_e at step k to merge with subgraph S_s. The parameters q_0, q are used in the same way as in the expansion transition rule. In the case $q > q_0$, we simply select the subgraph S_e with the maximal value of $\overline{\tau}_{is}\overline{\eta}_i$, 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. 22.

4.3 Local and Global Update Rules

The final step in defining the ACO method is specifying the global and local update rules. The proposed ACO algorithm corresponds to the ant colony system, in which only the best found solution deposits pheromone after each iteration of the colony. The global update rule can formally be defined using the following equations

$$\Delta\tau = Val(\Pi') \tag{23}$$

$$\tau_i = (1-p)\tau_{vs} + p\Delta\tau \qquad , \forall (v,s) \in \Pi' \tag{24}$$

In Eq. 23 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, which is defined in the implementation subsection. In Eq. 24, the parameter $p \in (0,1)$ is used to specify the impact of the global update rule. It is important to point out

that Eq. 24 only effects the values of pheromone τ_{vs} for $(v, s) \in \Pi'$ as in Eq. 15.

Next we define the local update rule, which is used to diversify the search of the solution space. In general this is done by decreasing values in the pheromone matrix, after an ant has performed the transition rule. Due to the way the elements of a solution are defined, in the application of ACO for MPGSD-LC, this rule can only be applied after an ant has generated a complete solution. In case ant i has generated a solution Π_i, the corresponding local update rule is defined using the following formula

$$\tau_{vs} = \varphi\tau_{vs} \qquad ,\forall(v, s) \in \Pi_i \tag{25}$$

In Eq. 25 $\varphi \in (0, 1)$ is used to specify the influence of the local update rule.

4.4 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 function is necessary for both the initialization of the pheromone matrix and for the global update rule. It is defined as follows

$$T = \sum_{v \in V_s} sup(v) \tag{26}$$

$$Val(\Pi) = \frac{1}{T - D(\Pi) + 1} \tag{27}$$

As given in Eq. 27, the quality of a solution $Val(\Pi)$ is inversely proportional to the difference of T, the total initially available supply in G, and the satisfied demand $D(\Pi)$. With the goal of avoiding division by zero, one is added since there is a possibility of $T = D(\Pi)$. The pheromone matrix is initialized by setting the value of each of its elements τ_{vs} to $Val(\Pi_g)$, where Π_g is the solution acquired using the presented greedy algorithm. To be exact, in the expansion stage the node selection heuristic is the maximal demand and the subgraph selection heuristic is the maximal available supply, as in (Jovanovic and Bousselham, 2014). In the merger stage, the heuristic functions are given in Eqs. 13, 14. With the intention of having a comprehensive presentation, we detail the proposed ACO algorithm in the form of the pseudo code shown in Algorithm 3.

As it can be observed from the pseudo code, the algorithm starts by generating a solution using the greedy algorithm. Next, we initialize the pheromone matrix τ using this solution. The main loop corresponds to one iteration of the colony in which each of the n artificial ants generates a solution. The next loop is used to generate a solution for a single ant. In this loop, first an initial partitioning Π is created. As in the case of the greedy algorithm the complete solution is generated by consecutive expansion and merger stages which are given in two separate loops. In practice the method for generating an individual solution in the ACO is the same as in the case of the greedy method

Algorithm 3 ACO

```
Generate solution Π_g using the greedy algorithm
Initialize the pheromone matrix τ with Val(Π_g)

while (Maximal number of iterations not reached) do
  for all n ants do
    Initialize Π = {S_1, .., S_n} , S_i = {s_i}

    repeat
      while (Sum(|N_i|) > 0) do
        Randomly select S_i, where |NV(S_i)| > 0
        Select v for S_i using transition rule
        S_i = S_i ∪ v
        Update auxiliary structures
      end while

      while (Sum(|NS_i|) > 0) do
        Select S_i using h_se(S_i)
        Select S_j ∈ NS_i using transition rule
        S_i = S_i ∪ S_j
        Π = Π \ {S_j}
        Update auxiliary structures
      end while
    until (NoChange)

    Set Π to N subgraphs with maximal values of sup_i
    Apply local update rule for Π
  end for
  Apply global update rule for Π_best
end while
```

with some important differences. First in the case of the expansion stage, the subgraph that is being expanded is selected randomly from the set of available ones. For both, the expansion and the merger stage the transition rule is used instead of the heuristic function. For the algorithm to be computationally efficient it is recommended to include some auxiliary structures similar to the ones presented in (Jovanovic and Bousselham, 2014). Such structures, generally, need to be updated after each change in the partial solution. The local update rule is applied after an ant has acquired a complete solution using Eq. 25. After all of the ants in the colony have generated their solutions we apply the global update rule given in Eq. 24 for the best solution Π_{best} found by the algorithm for all the previous iterations.

5 Results

In this section we present the results of our computational experiments used to evaluate the performance of the proposed ACO method. To be able to observe the effect of the use of the ACO meta-heuristic we give a comparison with the presented greedy algorithm (Gr). Both algorithms have been implemented in C# using Microsoft Visual Studio 2012. The source code and the executive

files have been made available at Jovanovic (2015). 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.

5.1 Benchmark Data

Due to the fact that the problem of MPGSD-LC has not previously been explored, before presenting the results of our computational experiments we first give a detailed description of the method for generating test instances. We have generated separate sets of problem instances for general graphs and trees. With the goal of having an extensive set of test problems a wide range of graph sizes has been considered. The generated test instances have 10-100 supply nodes and 30-1000 demand nodes. For each pair (ns, nd), number of supply and demand nodes, problem instances having a maximal number of subgraphs $nsub = 3, 5, 10$ have been generated with the constraint that $ns/nsub > 1$. For each triplet $(ns, nd, nsub)$, 40 different problem instances have been created using different seeds for the random generator using the following algorithm.

The first step was generating $ns + nd$ random positive integer numbers, corresponding to node weights, with a uniform distribution within the interval $[-10, -39]$. General graphs had a total of $(n + m) * 2$ random edges, with the constraint that the graph had to be connected. In case of the second type of graphs, i.e. trees, we would simply generate a random tree for $ns + nd$ nodes.

For both types of graphs, for a problem with a maximal number of subgraphs $nsub$, the next step was to select $nsub$ random nodes as seeds for the subgraphs (partitions). The subgraphs are grown using an iterative method until all the nodes of the original graph are contained in one of the subgraphs. The growth of subgraph S_i has been performed by expanding it to a random neighboring node that does not belong to any of the other subgraphs.

The number of supply nodes ns_i in each of the subgraph S_i was generated using the following iterative procedure. Initially all $ns_i = 1$. At each iteration a random graph S_i is selected and for it ns_i is incremented by one if $ns_i < |S_i|/2 - 1$. The next step was randomly selecting ns_i nodes inside S_i which will be supply nodes. The total supply w_i in such partition would be calculated using the following formula

$$w_i = \sum_{a \in S_i} sup(a) - \sum_{v \in Sel_i} sup(v) \tag{28}$$

Eq. 28 states that the total supply inside of partition S_i is equal to the sum of the weights of all nodes inside S_i minus the sum of the weights of all the nodes Sel_i selected to be supply nodes. The following step was distributing the total supply among the nodes that have been selected to be supply nodes. In practice this means that we randomly generate ns_i numbers having the sum w_i. This has been done by generating $ns_i - 1$ distinct random integer numbers between 0 and $w_i - 1$. These numbers are put in an array A with the addition of 0 and w_i, and sorted. The supply corresponding to the i-th node was equal

to $A[i+1] - A[i]$. The final step was setting the maximal allowed supply in a subgraph M_S to the maximal value of w_i.

For problem instances generated using the proposed method the optimal solution is known and is equal to the sum of supplies of all the supply nodes. It is important to mention that by using the proposed method for generating problem instances it is possible to have partitionings that do not have $nsub$ subgraphs with demand nodes. This is due to the possibility that some seed nodes may be cut off from the rest of the graph. Such partitionings have been excluded from the test data sets. The generated test instances are available for download at
`http://mail.ipb.ac.rs/~rakaj/home/graphsdlc.htm`.

5.2 Computational Experiments

The goal of the computational experiments was to evaluate the performance of both the proposed greedy and ACO algorithms. The tests have been done on 33 different problem sets defined using triplets $(ns, nd, nsub)$ for each type of graphs (general and trees). Each such problem set had 40 different problem instances. The ACO method was defined using the following parameters. The colony had 10 artificial ants and 150 iterations of the algorithm were performed. In practice this means that 1500 solutions have been generated for each test instance. The parameters for defining global and the local update rules had the following values $p = 0.1$ and $\varphi = 0.9$. The value $q_0 = 0.1$ was used for defining the exploitation/exploration rate. The chosen values for p, φ, q_0, n correspond to the ones commonly used in ACO implementations. In our initial tests we have observed that they produce the best performance. For each problem instance only a single run of the ACO algorithm has been performed.

The results of the conducted computational experiments are given in Tables 1 and 2 for general graphs and trees, respectively. The values in these tables are given in relation to each set of 40 different problem instances. To be exact, we present the average normalized error of the found solutions compared to the known optimal ones. The normalized error is calculated as $(Optimal - found)/Optimal*100$, and we show the average values for the 40 test instances. To be able to observe the robustness of the approach, Tables 1 and 2 also include the standard deviation and maximal normalized errors. For both graph types, the number of found optimal solutions (hits) for each problem set are also presented. Finally, to evaluate the scaling of the approach we give the average calculation time of the method. Average computational time is equal to the total calculation for all the problem instances in one problem set divided by 40.

From the results in Tables 1, 2, we first notice that the performance of the greedy algorithm has been highly dependent on individual problem instances. While the average error was 6-12.5%, and 11-18.5% for general graphs and trees, respectively, the maximal error was significant in both cases. To be more

Table 1 Comparison of the proposed algorithms for general graphs when 1500 solutions have been generated for ACO.

Sup X Dem X Sub	*Avg*(*Stdev*)		*Max*		*Hits*		Time(s)
	Gr	ACO	Gr	ACO	Gr	ACO	ACO
3 Subgraphs							
10 X 30	7.87(5.54)	1.58(1.70)	22.22	10.80	0	8	7.25e-3
10 X 50	7.67(4.68)	0.62(0.41)	18.50	1.24	0	12	3.77e-2
10 X 100	5.96(4.75)	0.32(0.17)	16.33	0.44	0	9	1.34e-1
25 X 75	9.37(4.98)	0.37(0.29)	21.01	0.77	0	15	1.59e0
25 X 125	10.17(5.13)	0.23(0.17)	19.42	0.51	0	14	3.08e0
25 X 250	11.15(4.50)	0.12(0.08)	21.05	0.19	0	11	5.24e0
50 X 150	9.94(5.91)	0.27(0.15)	26.12	0.68	0	7	7.63e0
50 X 250	9.76(5.22)	0.21(0.13)	21.00	0.79	0	2	1.11e1
50 X 500	11.31(4.65)	0.14(0.08)	20.46	0.40	0	0	2.11e1
100 X 300	10.61(5.24)	0.33(0.22)	24.17	0.88	0	0	3.12e1
100 X 500	13.80(5.34)	0.33(0.32)	23.99	1.33	0	0	4.58e1
100 X 1000	12.49(5.75)	0.47(0.47)	24.24	1.80	0	0	8.41e1
5 Subgraphs							
10 X 30	9.53(4.70)	2.10(0.80)	20.51	4.82	0	1	1.25e-3
10 X 50	7.74(3.69)	1.03(0.36)	19.42	1.86	0	2	3.16e-2
10 X 100	5.41(3.28)	0.48(0.17)	15.38	1.25	0	0	1.08e-1
25 X 75	9.44(4.33)	0.70(0.29)	16.53	1.57	0	2	1.60e0
25 X 125	8.13(3.87)	0.41(0.19)	18.13	1.22	0	2	3.04e0
25 X 250	6.94(3.95)	0.26(0.11)	16.87	0.55	0	0	5.04e0
50 X 150	9.09(5.16)	0.51(0.35)	20.60	1.91	0	0	7.39e0
50 X 250	9.06(4.22)	0.36(0.17)	25.49	0.88	0	0	1.10e1
50 X 500	8.30(3.91)	0.34(0.19)	19.10	0.95	0	0	2.05e1
100 X 300	8.74(3.68)	0.71(0.49)	15.77	2.48	0	0	2.95e1
100 X 500	10.14(4.22)	0.65(0.35)	22.21	1.51	0	0	4.36e1
100 X 1000	10.09(4.28)	0.71(0.54)	19.01	2.61	0	0	7.95e1
10 Subgraphs							
25 X 75	8.32(2.63)	1.48(0.63)	13.36	2.83	0	0	1.58e0
25 X 125	7.25(2.46)	0.92(0.27)	13.01	1.50	0	0	2.90e0
25 X 250	6.06(3.01)	0.74(0.23)	15.10	1.16	0	0	4.98e0
50 X 150	7.37(2.54)	1.10(0.42)	13.56	2.52	0	0	7.06e0
50 X 250	8.33(3.75)	1.00(0.39)	21.23	2.07	0	0	1.09e1
50 X 500	7.25(2.11)	0.72(0.26)	11.44	1.54	0	0	2.04e1
100 X 300	7.29(2.73)	1.15(0.44)	13.55	2.63	0	0	2.85e1
100 X 500	8.23(2.86)	1.16(0.44)	17.13	2.76	0	0	4.24e1
100 X 1000	8.40(2.94)	0.96(0.43)	14.60	2.06	0	0	7.76e1

specific, the maximal error was generally around 15-25% for general graphs and 30-40% for trees. The proposed ACO algorithm gave a significant improvement in the quality of found solutions, with an average error of 0.3-1.5% in case of general graphs and 1-2.5% for trees. It is interesting that the performance of the ACO method was much more consistent in the case of general graphs,

Table 2 Comparison of the proposed algorithms for trees when 1500 solutions have been generated for ACO.

Sup X Dem X Sub	*Avg*(*Stdev*)		*Max*		*Hits*		Time(s)
	Gr	ACO	Gr	ACO	Gr	ACO	ACO
3 Subgraphs							
10 X 30	13.69(10.65)	2.51(4.08)	37.79	21.44	4	21	3.25e-3
10 X 50	12.35(11.00)	0.93(2.27)	40.78	12.50	6	29	1.20e-2
10 X 100	13.60(10.28)	0.62(1.29)	36.81	6.22	3	27	5.32e-2
25 X 75	15.61(11.51)	1.68(2.18)	37.83	10.07	2	16	1.40e0
25 X 125	14.93(11.02)	0.94(2.52)	41.84	13.46	1	25	1.71e0
25 X 250	13.73(9.67)	1.22(3.22)	41.73	13.99	0	27	3.67e0
50 X 150	11.77(8.35)	2.07(3.89)	32.81	14.82	0	17	5.62e0
50 X 250	15.36(10.77)	0.73(1.39)	38.15	7.11	1	18	8.86e0
50 X 500	16.70(10.69)	1.36(2.54)	39.27	10.96	0	19	1.61e1
100 X 300	18.27(10.12)	1.62(2.16)	39.68	9.80	0	11	2.33e1
100 X 500	16.94(9.57)	1.19(2.18)	36.24	10.07	0	13	3.48e1
100 X 1000	15.65(10.51)	1.42(2.53)	37.34	12.35	1	16	6.83e1
5 Subgraphs							
10 X 30	14.23(9.31)	2.71(3.85)	51.20	19.21	4	19	4.50e-3
10 X 50	12.60(7.76)	1.70(3.40)	32.40	12.57	1	28	1.75e-3
10 X 100	14.25(9.13)	0.78(1.97)	35.27	10.35	0	29	5.12e-2
25 X 75	13.53(9.49)	1.64(2.28)	40.10	10.37	3	15	1.20e0
25 X 125	16.51(7.99)	1.20(1.76)	37.88	5.93	0	16	1.62e0
25 X 250	15.17(10.45)	0.61(1.03)	39.53	4.86	0	24	3.40e0
50 X 150	15.60(9.42)	1.90(2.55)	42.96	10.48	1	11	5.11e0
50 X 250	13.60(8.97)	1.08(1.55)	36.42	5.30	0	17	8.03e0
50 X 500	17.29(9.54)	1.14(2.17)	40.37	11.13	0	14	1.46e1
100 X 300	20.27(9.24)	2.54(2.88)	37.68	13.13	0	8	2.25e1
100 X 500	15.67(8.34)	1.42(1.63)	30.24	5.62	0	10	3.28e1
100 X 1000	17.69(9.67)	2.48(2.87)	41.84	11.50	0	4	5.89e1
10 Subgraphs							
25 X 75	13.88(8.53)	1.89(1.69)	34.67	6.60	0	8	7.28e-1
25 X 125	11.96(6.47)	1.60(1.77)	25.69	7.58	2	6	1.61e0
25 X 250	17.50(8.30)	0.89(1.12)	40.43	4.06	0	16	3.21e0
50 X 150	12.26(5.40)	1.95(1.75)	27.24	7.64	0	5	4.81e0
50 X 250	12.32(6.65)	1.32(1.45)	28.36	6.37	0	8	7.70e0
50 X 500	13.09(5.76)	1.05(1.32)	24.41	4.47	0	10	1.38e1
100 X 300	12.82(5.86)	2.23(1.72)	25.96	7.39	0	1	2.11e1
100 X 500	11.07(4.87)	1.38(1.49)	21.01	5.02	0	6	3.02e1
100 X 1000	14.01(6.67)	1.09(1.02)	29.46	4.22	0	5	5.47e1

where the standard deviation was small and the maximal error was generally around 0.5-3%. For trees, the maximal error was generally around 5-15%, while having higher values for smaller graphs. On the other hand, the ACO algorithm managed to find optimal solutions in more than 30% of the problem instances for trees, and only in around 7% of general graphs. The ACO method generally

performed worst when $ns/nsub$ was smallest. The computational time of the algorithm was more dependent on the number of supply than demand nodes. The calculation time was not significantly effected by the maximal number of allowed subgraphs in a partitioning.

With the goal of having a better evaluation of the proposed ACO method, it has been compared to the GRASP (Feo and Resende, 1995) presented above. We have also explored the effect of adding the local search to the ACO algorithm (ACO-C). To be more specific, in the ACO algorithm the local search has been applied to every solution that has been generated by the artificial ants. The chosen local search was the same as the one previously developed for the MPGSD (Jovanovic et al, 2015), which is also used for the GRASP. In case of GRASP and ACO-C we have generated 1500 different solutions. The results of these tests can be seen in Tables 3 and 4 for general graphs and trees, respectively. As in the case of the comparison of ACO and the greedy method, we compare the average error, maximal error and number of found optimal solutions. The computational times are not included in Tables 3 and 4, since there are very similar, with ACO being some 20-30% faster than the other two methods due to the fact that no local search is used.

From the results in Tables 3 and 4 we can first notice that the GRASP gives a very significant improvement when compared to the basic greedy algorithm, generally having an average error 0.5-2.5% for general graphs and 2.5-6.0% in case of trees. On the other hand, GRASP manages to achieve better results than the ACO method only for a few of the smallest graph sizes. Overall the ACO significantly outperforms GRASP, which indicates the effectiveness of using the pheromone trail for guiding the exploration of the solution space. The addition of the local search to ACO significantly improves the quality of the found solutions. The ACO-C had generally an average error between 0.01-1% and 0.3-2% for general graphs and trees, respectively. The most notable improvement of ACO-C compared to ACO is in the number of found optimal solutions which has risen to 28% and 50% for general graphs and trees, respectively. It is important to note that although ACO-C significantly outperforms ACO, for a few graphs sizes it produced worse average errors than the basic algorithm.

6 Conclusion

In this paper a new version of the MPGSD was presented which is more suitable for potential applications in the field of interconnected microgrids. Approximate solutions for the newly defined problem, MPGSD-LC, are found using a two stage greedy algorithm. The presented greedy method is used as a basis for an ACO algorithm. The proposed ACO method introduces a novel approach of using the pheromone trail by separating the pheromone matrix and the method for extracting values from it. In this way it was possible to use the same pheromone matrix for both stages of the proposed ACO method.

Our computational experiments have shown that the proposed approach is very suitable for the problem of interest. It was a significant improvement compared to the basic greedy algorithm, and managed to find optimal solutions for many of the test problem instances. The average relative error was never greater than 3%, and in nearly half the cases it was less than 1%. The tests have been performed on trees and general graphs. When we speak about optimality, this is based on the properties of the data generation routine that we have proposed, guaranteeing knowledge about the optimal objective function values.

In the future we plan to extend this research in two main directions. First, we plan to adapt the method to a stochastic version of the problem. The other direction is the development of a method that would be suitable for solving very large problem instances. 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.

Acknowledgements The comments of two anonymous referees are greatly appreciated.

References

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

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

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

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

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

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

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

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

Feo TA, Resende MG (1995) Greedy randomized adaptive search procedures. Journal of Global Optimization 6(2):109–133

Hart JP, Shogan AW (1987) Semi-greedy heuristics: An empirical study. Operations Research Letters 6:107–114

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

Hinne M, Marchiori E (2011) Cutting graphs using competing ant colonies and an edge clustering heuristic. In: Merz P, Hao JK (eds) Evolutionary Computation in Combinatorial Optimization, Lecture Notes in Computer Science, vol 6622, Springer, Berlin, pp 60–71

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

Ito T, Demaine ED, Zhou X, Nishizeki T (2008) Approximability of partitioning graphs with supply and demand. Journal of Discrete Algorithms 6(4):627 – 650

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

Jovanovic R (2015) Benchmark data sets for the problem of partitioning graphs with supply and demand. URL `http://mail.ipb.ac.rs/~rakaj/home/graphsdlc.htm`

Jovanovic R, Bousselham A (2014) 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, pp 154–159

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

Jovanovic R, Tuba M (2013) Ant colony optimization algorithm with pheromone correction strategy for the minimum connected dominating set problem. Computer Science and Information Systems 10(3):133–149

Jovanovic R, Tuba M, Voß S (2014) An ant colony optimization algorithm for partitioning graphs with supply and demand URL `http://arxiv.org/abs/1411.1080`, `1503.00899`

Jovanovic R, Bousselham A, Voß S (2015) A heuristic method for solving the problem of partitioning graphs with supply and demand. Annals of Operations Research DOI 10.1007/s10479-015-1930-5

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

Morishita S, Nishizeki T (2013) Parametric power supply networks. In: Du DZ, Zhang G (eds) Computing and Combinatorics, Lecture Notes in Computer Science, vol 7936, Springer, Berlin, pp 245–256

Narayanaswamy NS, Ramakrishna G (2012) Linear time algorithm for tree t-spanner in outerplanar graphs via supply-demand partition in trees URL `http://arxiv.org/abs/1210.7919`, accepted in Discrete Applied Mathematics (2014)

Popa A (2013) Modelling the power supply network - hardness and approximation. In: Chan TH, Lau L, Trevisan L (eds) Theory and Applications of Models of Computation, Lecture Notes in Computer Science, vol 7876, Springer, Berlin, pp 62–71

Reinelt G, Wenger KM (2010) Generating partitions of a graph into a fixed number of minimum weight cuts. Discrete Optimization 7(12):1 – 12

Reinelt G, Theis DO, Wenger KM (2008) Computing finest mincut partitions of a graph and application to routing problems. Discrete Applied Mathematics 156(3):385 – 396

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

Voß S, Fink A, Duin C (2005) Looking ahead with the pilot method. Annals of Operations Research 136(1):285–302

Table 3 Comparison of the proposed algorithms with the GRASP for general graphs when 1500 solutions have been generated for ACO (best results are underlined).

Sup X Dem X Sub	*Avg*(*Stdev*)			*Max*			*Hits*		
	GRASP	ACO	ACO-C	GRASP	ACO	ACO-C	GRASP	ACO	ACO-C
3 Subgraphs									
10 X 30	0.95(1.76)	1.58(1.70)	0.89(1.74)	10.17	10.80	10.49	23	8	23
10 X 50	0.21(0.41)	0.62(0.41)	0.10(0.27)	1.59	1.24	0.88	31	12	35
10 X 100	0.06(0.33)	0.32(0.17)	0.01(0.06)	2.05	0.44	0.42	38	9	39
25 X 75	0.64(0.78)	0.37(0.29)	0.14(0.24)	3.90	0.77	0.66	13	15	30
25 X 125	0.50(0.69)	0.23(0.17)	0.01(0.05)	3.97	0.51	0.32	12	14	39
25 X 250	1.06(1.52)	0.12(0.08)	0.02(0.06)	0.19	6.34	0.17	11	5	34
50 X 150	0.96(1.06)	0.27(0.15)	0.07(0.12)	4.96	0.68	0.28	5	7	29
50 X 250	0.89(1.12)	0.21(0.13)	0.07(0.09)	5.46	0.79	0.25	1	2	25
50 X 500	1.33(0.94)	0.14(0.08)	0.10(0.07)	3.90	0.40	0.33	0	0	6
100 X 300	1.62(1.43)	0.33(0.22)	0.24(0.24)	6.03	0.88	1.43	0	0	4
100 X 500	1.37(1.26)	0.33(0.32)	0.25(0.34)	5.65	1.33	1.41	0	0	5
100 X 1000	2.57(2.41)	0.47(0.47)	0.32(0.37)	11.08	1.80	1.82	0	0	3
5 Subgraphs									
10 X 30	1.16(0.78)	2.10(0.80)	1.19(0.69)	2.79	4.82	2.62	11	1	9
10 X 50	0.72(0.52)	1.03(0.36)	0.59(0.39)	2.39	1.86	1.01	10	2	12
10 X 100	0.39(0.47)	0.48(0.17)	0.20(0.21)	3.03	1.25	0.58	10	0	21
25 X 75	0.81(0.51)	0.70(0.29)	0.36(0.27)	2.39	1.57	0.64	3	2	14
25 X 125	0.68(0.62)	0.41(0.19)	0.24(0.15)	3.68	1.22	0.37	3	2	11
25 X 250	0.82(0.79)	0.26(0.11)	0.12(0.07)	3.04	0.55	0.17	0	0	10
50 X 150	1.02(0.65)	0.51(0.35)	0.24(0.26)	3.01	1.91	1.68	0	0	10
50 X 250	1.23(0.84)	0.36(0.17)	0.21(0.14)	4.30	0.88	0.68	0	0	3
50 X 500	1.49(0.91)	0.34(0.19)	0.17(0.11)	4.37	0.95	0.48	0	0	2
100 X 300	1.95(1.07)	0.71(0.49)	0.44(0.39)	4.81	2.48	1.93	0	0	2
100 X 500	2.04(0.88)	0.65(0.35)	0.38(0.26)	3.82	1.51	1.06	0	0	0
100 X 1000	2.69(1.16)	0.71(0.54)	0.63(0.48)	5.82	2.61	2.03	0	0	0
10 Subgraphs									
25 X 75	1.35(0.55)	1.48(0.63)	0.71(0.27)	2.94	2.83	1.60	0	0	0
25 X 125	1.00(0.54)	0.92(0.27)	0.37(0.09)	2.36	1.50	0.71	0	0	0
25 X 250	0.94(0.65)	0.74(0.23)	0.27(0.12)	2.97	2.83	0.69	0	0	0
50 X 150	1.63(0.85)	1.10(0.42)	0.50(0.24)	4.10	2.52	1.15	0	0	0
50 X 250	1.80(0.83)	1.00(0.39)	0.44(0.24)	4.48	2.07	1.16	0	0	0
50 X 500	1.93(0.76)	0.72(0.26)	0.38(0.20)	4.51	1.54	0.99	0	0	0
100 X 300	1.84(0.74)	1.15(0.44)	0.67(0.26)	3.97	2.63	1.38	0	0	0
100 X 500	2.42(0.85)	1.16(0.44)	0.79(0.37)	3.94	2.76	1.68	0	0	0
100 X 1000	2.92(1.05)	0.96(0.43)	0.84(0.48)	5.47	2.06	2.05	0	0	0

Table 4 Comparison of the proposed algorithms with the GRASP for tree graphs when 1500 solutions have been generated for ACO (best results are underlined).

Sup X Dem X Sub	$Avg(Stdev)$			Max			$Hits$		
	GRASP	ACO	ACO-C	GRASP	ACO	ACO-C	GRASP	ACO	ACO-C
3 Subgraphs									
10 X 30	5.19(9.13)	2.51(4.08)	1.44(4.07)	37.48	21.44	21.44	25	21	33
10 X 50	3.18(6.40)	0.93(2.27)	1.05(3.48)	25.70	12.50	17.80	26	29	34
10 X 100	2.45(4.65)	0.62(1.29)	0.26(0.71)	17.49	6.22	3.31	24	27	34
25 X 75	5.48(7.43)	1.68(2.18)	1.58(2.59)	30.97	10.07	10.44	12	16	24
25 X 125	3.76(6.17)	0.94(2.52)	0.78(2.27)	26.07	13.46	9.79	12	25	32
25 X 250	3.08(4.93)	1.22(3.22)	0.92(2.71)	20.05	13.99	13.99	18	27	31
50 X 150	2.95(4.11)	2.07(3.89)	2.13(4.04)	14.83	14.82	15.85	8	17	22
50 X 250	3.15(4.95)	0.73(1.39)	0.66(1.38)	22.42	7.11	6.23	13	18	24
50 X 500	5.58(7.95)	1.36(2.54)	1.51(2.82)	29.64	10.96	10.96	9	19	23
100 X 300	3.03(5.22)	1.62(2.16)	1.51(2.17)	23.96	9.80	8.19	10	11	17
100 X 500	2.05(3.42)	1.19(2.18)	0.87(1.81)	12.20	10.07	9.08	8	13	19
100 X 1000	3.71(6.24)	1.42(2.53)	1.35(4.81)	28.43	21.35	16.12	6	16	17
5 Subgraphs									
10 X 30	2.50(4.02)	2.71(3.85)	0.65(1.50)	17.22	19.21	5.29	22	19	33
10 X 50	2.59(4.00)	1.70(3.40)	1.34(3.12)	12.29	12.57	12.57	25	28	31
10 X 100	1.69(2.91)	0.78(1.97)	0.24(0.69)	9.67	10.35	3.32	26	29	34
25 X 75	4.15(4.21)	1.64(2.28)	1.34(2.50)	14.81	10.37	10.58	9	15	24
25 X 125	6.27(6.38)	1.20(1.76)	1.08(2.40)	23.55	5.93	13.65	10	16	22
25 X 250	5.75(6.81)	0.61(1.03)	0.89(2.10)	30.38	4.86	11.24	10	24	25
50 X 150	4.97(6.98)	1.90(2.55)	1.85(2.84)	36.84	10.48	10.01	7	11	17
50 X 250	4.57(4.44)	1.08(1.55)	1.26(1.89)	18.95	5.30	7.88	7	17	19
50 X 500	5.13(5.72)	1.14(2.17)	1.15(2.16)	22.79	11.13	10.76	2	14	17
100 X 300	4.24(4.59)	2.54(2.88)	2.15(2.73)	17.55	13.13	12.47	3	8	8
100 X 500	4.63(4.84)	1.42(1.63)	1.55(1.79)	22.09	5.62	6.63	3	10	10
100 X 1000	7.12(6.71)	2.48(2.87)	1.70(2.37)	28.32	11.50	9.31	0	4	6
10 Subgraphs									
25 X 75	4.21(4.37)	1.89(1.69)	0.61(0.75)	20.37	6.60	3.07	7	8	20
25 X 125	3.76(4.13)	1.60(1.77)	1.05(1.60)	17.60	7.58	6.51	8	6	18
25 X 250	3.86(4.11)	0.89(1.12)	1.04(1.30)	17.00	4.06	5.24	8	16	20
50 X 150	4.04(2.93)	1.95(1.75)	1.34(1.24)	10.50	7.64	4.18	3	5	10
50 X 250	4.31(4.20)	1.32(1.45)	1.14(1.28)	21.13	6.37	5.12	2	8	14
50 X 500	5.27(3.66)	1.05(1.32)	0.91(1.24)	15.39	4.47	4.50	1	10	18
100 X 300	4.55(3.23)	2.23(1.72)	2.04(1.71)	12.16	7.39	7.54	1	1	5
100 X 500	4.32(3.46)	1.38(1.49)	1.10(1.23)	15.12	5.02	3.94	0	6	9
100 X 1000	5.99(4.34)	1.09(1.02)	1.30(1.38)	22.26	4.22	4.85	0	5	8