Applicable Analysis and Discrete Mathematics
available online at http://pefmath.etf.rs

Appl. Anal. Discrete Math. **14** (2020), 775–799.
https://doi.org/10.2298/AADM190227052J

A MATHEURISTIC APPROACH FOR SOLVING THE 2-CONNECTED DOMINATING SET PROBLEM

Raka Jovanovic and Stefan Voß*

This paper describes a matheuristic approach for solving the 2-connected dominating set problem (2-CDS). The goal of the proposed method is to find near optimal solutions for large graphs. The algorithm is based on a Greedy Randomized Adaptive Search Procedure (GRASP). Its performance is enhanced by combining it with a second local search method that uses a Mixed Integer Program. The performance of the proposed method is evaluated on large unit disc graphs having varying edge densities and on general graphs.

1. INTRODUCTION

A dominating set for a graph $G(V, E)$, where V represents the set of vertices of the graph and E the set of edges of the graph, is a subset of vertices $D \subseteq V$ that has the property that every vertex in G either belongs to D or is adjacent to a vertex in D. Finding the dominating set with the smallest possible cardinality among all dominating sets, for a graph, is one of the standard NP-hard problems [**8**]. In this work, we focus on the problem of the 2-connected dominating set (2-CDS) which has the additional restriction that there are two vertex-disjoint paths between any two nodes $n, m \in D$. These paths consist only of nodes that are elements of D. An example of a problem instance and solution for the 2-CDS can be seen in Fig. 1.

The Dominating Set Problem (DSP) has been extensively explored due to its close connection with wireless networks [**21, 28**] and sensor networks [**28**]. Due to the importance of the DSP, the problem has been solved using a wide range

*Corresponding author. Raka Jovanovic
2010 Mathematics Subject Classification. 68W20, 68T20.
Keywords and Phrases: Matheuristic, Dominating Set Problem, GRASP.

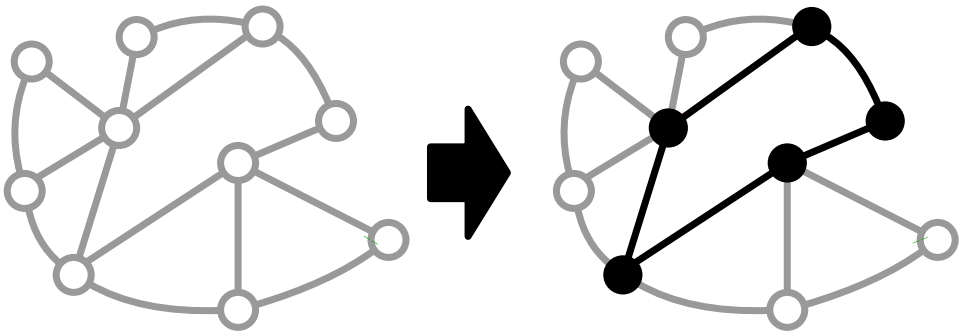

Figure 1: Example of a problem instance (left) and solution (right) for the 2-CDS. In the solution the filled black nodes represent a minimal 2-connected dominating set.

of metaheuristic methods like genetic algorithms [**10**] and ant colony optimization [**11**]. Several variations have been considered like the weighted [**17**] and connected version [**8, 9, 4**] of the problem. Different types of domination have also been considered in literature like the quasiperfect [**6**] and total Roman [**1**] ones. When modeling ad hoc wireless and sensor networks, using graphs, the Connected DSP (CDSP) gives us a significantly more realistic representation of such systems when compared with the basic DSP [**28**]. For the CDSP several different approaches have been used for finding either exact or approximate solutions [**8, 16, 28**].

In recent years there has been an increased interest in the 2-connected version of the problem since it provides a higher level of failure resistance of the modeled wireless networks [**26, 27, 22**]. A significant part of the published research has been dedicated to finding optimal solutions using Mixed Integer Programming (MIP). The 2-CDS is a restriction of the more general problem of finding the minimal k-connected m-dominating sets (k-m-CDS). Several methods have been developed for finding optimal and approximate solutions for the k-m-CDS and its restrictions. In the case of finding optimal solutions for this group of problems, MIP has proven to be very successful for medium-size graphs [**20, 4, 2**]. In case of solving larger problem instances of k-m-CDS and its restrictions several approximate methods have been developed. Some examples are constructive approaches [**7**] and exploiting the relationship between the maximal independent set and the general k-m-CDS [**25, 21, 7**] algorithms.

In this work, we focus on developing a method that manages to find near optimal solutions for the minimal 2-connected dominating set problem (2-CDS). Different approaches have been used to solve this restriction of the k-m-CDS, like the use of MIP [**22, 20**] and constructive algorithms [**27, 26**]. Contemporary research has shown that matheuristics, which combine metaheuristics and MIP, are very suitable for this type of task. This new type of hybridized algorithms have proven their value for a wide range of problems ranging from berth allocation [**19**], lot sizing [**5**], facility location [**18**], etc. An overview of different approaches for

developing such methods can be seen in [**24**].

In this article, a matheuristic approach is developed for the 2-CDS. The first reason for developing such an approach is the lack of an efficient algorithm for solving large scale problem instances. Secondly, MIP models have been extensively researched and have proven to be very suitable for solving the 2-CDS on small enough graphs. The general idea of our approach is to develop an efficient method for combining a low computational cost heuristic based local search procedure with a high cost one based on MIP. Let us note that the use of MIP for a local search procedure has previously been successfully applied to the capacitated facility location problem [**18**]. The proposed method extends a Greedy Randomized Adaptive Search Procedure (GRASP) for the problem of interest. To be more precise, it uses the GRASP method for a global search, and MIP as a local search to further improve solutions generated in this way. In the proposed method, we also exploit the fact that good solutions frequently have a large level of similarity for lowering the number of times the MIP local search is applied.

In our computational experiments we compare the proposed method to MIP formulations and the basic GRASP algorithm. Our tests show that the proposed matheuristic approach manages to significantly outperform the previously developed GRASP algorithm, for the problem of interest, when both solution quality and computational time are considered. Further, it manages to be highly competitive to existing MIP formulations. One of the most important aspects of the proposed algorithm is that it can be used to significantly improve the performance of MIP methods for the 2-CDS.

The paper is organized as follows. In the next section we give a short outline of a MIP formulation for the 2-CDS from [**20**]. In Section 3, the previously developed greedy algorithm for the 2-CDS is presented. In the following section we describe the two local search procedures. In Section 5, implementation details of the proposed matheuristic approach are presented. After that we discuss the results of the performed computational experiments. The paper concludes in Section 7.

2. MIXED INTEGER PROGRAM

There is a wide range of potential MIP formulations for the 2-CDS out of which we have chosen the one based on cut sets as in [**20**], where an in depth analysis of the model can be found. In the formulation, we use the standard notation $\delta(S) = \{\{i, j\} \in E \mid i \in S \wedge j \in V \setminus S\}$ for all the edges containing only one node in S. For a given node $i \in V$, let Γ_i be the closed neighborhood of i, i.e., the subset of vertices of V which are adjacent to i. We also define $\Gamma_i^- = \Gamma_i \setminus \{i\}$. Further, let us define $\delta_k(S) = \{\{i, j\} \mid i \in S \setminus \{k\} \wedge j \in V \setminus (S \cup \{k\})\}$.

We will use the binary variables y_i, defined for $i = 1..|V|$, to state if a node is a part of the dominating set D. Next, auxiliary real variables x_e will be defined for all edges in $e \in E$. A variable x_e will state if an edge e is a part of the graph induced by the dominating set D. Then, in the model we minimize the following

objective function

$$\sum_{i \in V} y_i. \tag{1}$$

The mathematical model will contain the following constraints.

$$\sum_{j \in \Gamma_i^-} y_j \geq y_i + 1 \qquad i \in V \tag{2}$$

$$x_e \leq y_i \qquad e = \{i, j\} \in E \tag{3}$$

$$x_e \leq y_j \qquad e = \{i, j\} \in E \tag{4}$$

$$\sum_{e \in \delta(S)} x_e \geq 2(y_i + y_j - 1), \qquad S \subset V, i \in S, j \in V \setminus S \tag{5}$$

$$\sum_{e \in \delta_k(S)} x_e \geq y_i + y_j - 1, \qquad S \subset V \setminus \{k\}, i \in S, j \in V \setminus (S \cup \{k\}), k \in V \tag{6}$$

$$0 \leq x_e \leq 1 \qquad e \in E \tag{7}$$

$$0 \leq y_i \leq 1 \qquad i \in V \tag{8}$$

The constraint given in (2) insure the dominance property of set D and that for any $i \in D$ at least two of its neighbors are also in D. The constraints given in (3) and (4) are used to specify which edges will be a part of the graph induced using D. Next, the constraint given in (5) is used to guarantee the connectivity of set D. Finally, the constraint given in (6) is used to ensure that there are no cut vertices.

3. GREEDY ALGORITHM

In this section, we present a greedy method for generating solutions for the 2-CDS. The greedy method exploits the fact that any 2-connected graph has an open ear decomposition [**23**]. The algorithm consists in iteratively growing a 2-connected subgraph S by extending it with an open ear P. In practice this means that at each step of the algorithm, based on some heuristic function, a new ear P is selected from a suitably defined list of candidates. In the following text, we give an outline of the growth procedure which is also used in [**14, 15**] and the greedy algorithm which is similar to the one used for the 2-m-CDS in [**14**].

3.1 Growth procedure

Aiming at a comprehensive presentation, we start this subsection with the definition of an open ear decomposition. An ear of an undirected graph G is a path P where the two endpoints of the path may be the same, but no other edge

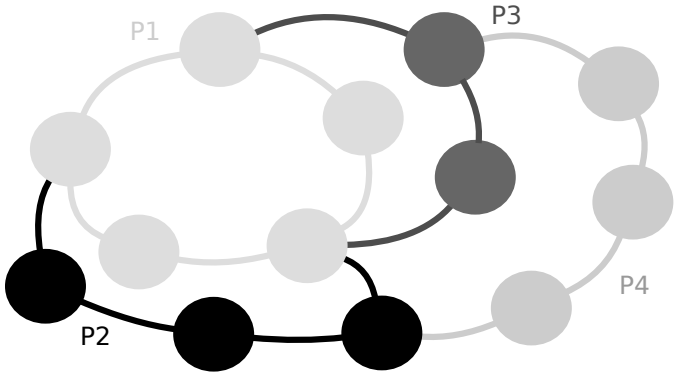

Figure 2: Examples of an open ear decomposition for a bi-connected graph. Different shades of grey are used for separate open ears.

or vertex appears more than once. In other words, any internal vertex of P has degree two in P. An open ear decomposition of graph G is a sequence of ears $P_0, \ldots, P_n$ in which only P_0 is a cycle. This sequence must satisfy the constraint that the endpoints of ear P_i belong to some P_j, P_k where $j, k < i$ where j, k are not necessarily distinct. No other vertices of P_i can belong to any P_j where $j < i$. An illustration of an open ear decomposition of a graph is given in Figure 2. The objective of the growth procedure is to produce an efficient way of generating a sequence $S_0 \subset S_1 \subset S_2 \subset \cdots$, where S_0 is an initial cycle and

$$S_{i+1} = S_i \cup P_i \tag{9}$$

where P_i is an open ear for S_i. From the construction procedure it is evident that each subgraph S_i will have an open ear decomposition, hence, it will be 2-connected.

As it has been shown in [**14, 15**] such a sequence can easily be generated by adapting the Breadth First Search (BFS). The basic idea is closely related to the way the BFS is used to find cycles in a graph. Let us assume that the root of the BFS tree is node r. During the BFS the first time a back-edge (u, v) is discovered an initial cycle S is found. This cycle consists of three segments: a BFS tree path from r to u, the back edge (u, v) and the BFS tree path from v to r. Let us assume that we continue the growth of the BFS tree and a new back-edge (s, t) is discovered, then an open ear is encountered if the following is satisfied. Let us define $root(u, P)$ as the node v which is the first ancestor of u, in the BFS tree, such that $v \in P$. In case $u \in P$, $root(u, P) = u$. The simplified notation $root(s)$ will be used if P is equal to the previously generated 2-connected subgraph S. If at least one of the nodes s, t is not in S and $root(s) \neq root(t)$ a new open ear P is found. More precisely, P consists of the following segments: the BFS tree path from $root(s)$ to s, the back edge (s, t) and the BFS tree path from t to $root(t)$. It is obvious that if we extend S with P, we can repeat this procedure and further grow a 2-connected subgraph.

There are two main differences in the adaption of the BFS used for implementing the growth procedure and the original BFS. First, in the original BFS, the node u which has the minimal distance to the root node r is first explored. Instead of this in the adaptation of the BFS the node u having the minimal distance $d(u)$ to the already generated 2-connected subgraph S is first explored. The second important difference is that for each node u, in the BFS tree, additional values $root(u)$, $d(u)$ need to be tracked. This is due to the fact that they can change when a new ear P is added to S.

Let us define $desc(u)$ as the set of all descendants of u in the BFS tree. In relation, we define an extension of this function to node sets P as $desc(P) = \bigcup_{i \in P} desc(u)$. The values of $root(u)$, $d(u)$ are calculated using an update procedure, performed after an ear P is added to S. More precisely, the following correction is applied to all $u \in P \cup desc(P)$:

$$root(v) = root(v, P). \tag{10}$$

$$d'(v) = \begin{cases} d'(v) - d'(root(v,P)) & v \notin P \\ 0 & v \in P \end{cases} \tag{11}$$

It is important to consider that the proposed correction for functions $d(u)$ will produce approximations to the exact distance $d'(u)$, which has the property that $d'(u) \geq d(u)$. This is due to the possible existence of an alternative path from u to S using some back edges which is shorter.

In the standard BFS, there is no change in the distance for visited nodes and no node is re-visited. In the proposed adaptation of BFS, such changes can occur and some revisits are necessary. The revisits are needed since some back edges that did not create open ears, may do so after the changes of the root values. Both of these issues are addressed simultaneously using the following approach. First, nodes will be re-added to the queue as a new ear is added to S and their re-evaluation is needed. Since it is possible for the same node to be added multiple times to the queue due to the addition of multiple ears, an additional value will be used to track if an evaluation is needed. The algorithm for growing a bi-connected subgraph is better understood by observing Algorithm 1.

The proposed algorithm starts with a standard BFS initialization of the distance, parent and descendant relations for all the nodes. We also add an additional property, which indicates if there is a need to evaluate a node. Initially the evaluation property $Eval$ will be set to $true$ for all nodes. An auxiliary structure is used to store all the properties of individual nodes, which can be accessed and updated using the node id. Some special initialization is needed for the root r and all its neighbors $N(r)$, for which details can be found in [**15**]. Next, all nodes in $N(r)$ are added to the queue Q. The main loop is executed for each node $current$ in Q until $Q = \emptyset$. For each such node, we first check if an evaluation is needed and if so, all its neighbors $N(current)$ are evaluated. For each $u \in N(current)$ we check if $(current, u)$ is a back-edge. In case this is not true, we add u to Q as in the BFS, and we set $root(u) = root(current)$. In case $(current, u)$ is a back-edge we check if it induces a new open ear P connected to S. If this holds, the subgraph

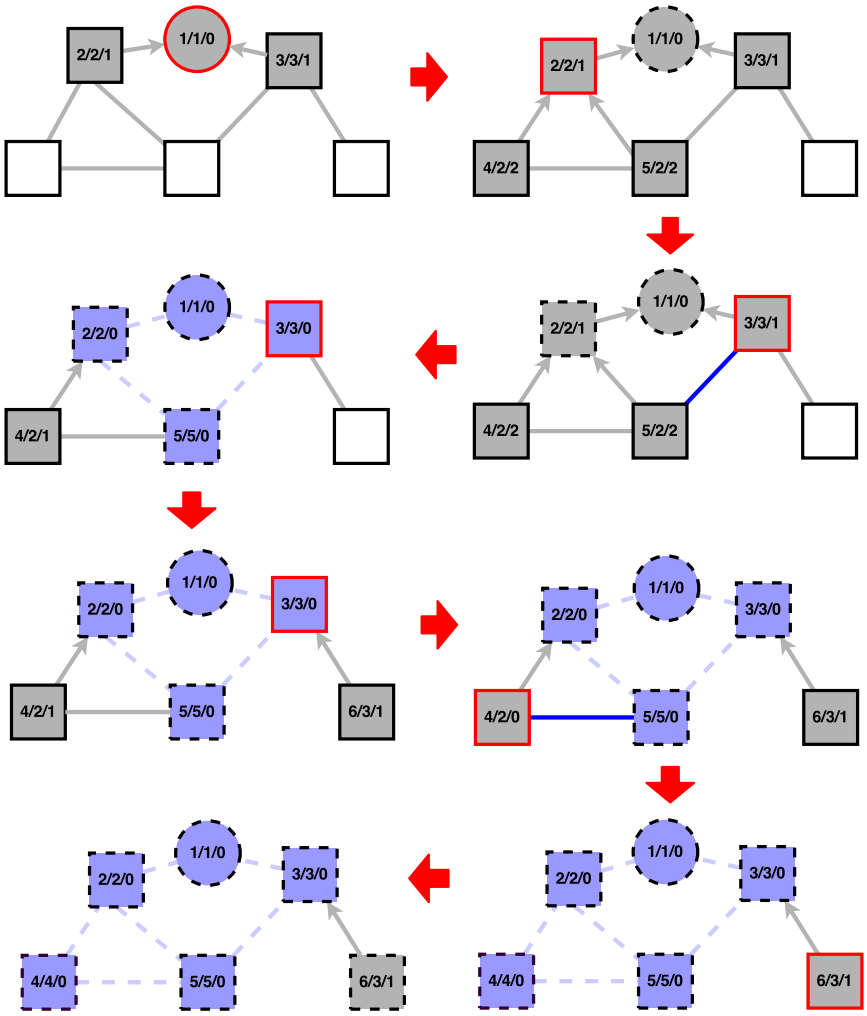

Figure 3: Illustration of steps in the growth procedure. The root node is represented with a circle and the rest of the nodes are shown as squares. Each node u stores 3 values, first is the id acquired by the BFS, the second is the value of $root(u)$ and the last is the values of $d(u)$. The node currently explored by the BFS is colored red. Gray arrows are used to indicate the structure of the BFS tree. Blue color edges are used for back-edges that are found at some step of the procedure. The violet color is used to show nodes that are a part of the found be bi-connected subgraph S and the corresponding edges are presented using violet dashed lines. Dashed borders of a node u indicate that the exploration of node u has been completed.

S is extended with P and necessary updates are performed. To be more precise,

Algorithm 1 Pseudo code for growing a 2-connected subgraph

```
For all u ∈ G initialize Dist, Eval, Parent
Perform initialization for r ∪ N(R)
Add all N(r) to Q
while (Q is not empty) ∧ (NotExitCriteria) do
    current = Q.dequeue()                    ▷ Using Queue (FIFO) structure
    if current.Eval then
        for all u ∈ N(current) do
            if (u, current) is BackEdge) then
                if u, current produce an open ear P then
                    S = S ∪ P
                    Perform necessary updates based on P
                end if
            else
                u.[Root, Dist] = [current.Root, current.Dist + 1]
                Update parent, child relations for u, current
                Q.enqueue(u)
            end if
        end for
        current.Eval = false
    end if
end while
```

for all $u \in P$ the distance is set to $d.Dist = 0$. Each node u now becomes a root of a new potential ear, so we set $u.root = u$. For each node $u \in P$ we wish to update the branch of the BFS tree whose root is u. All such nodes v are added to Q for re-evaluation and the values $v.root$, $v.Dist$ are corrected based on Eqs. (10), (11). Implementation details can be found in [**15**]. After all the elements of $N(current)$ are visited the evaluation of node *current* is complete and we set $current.Eval = false$. A graphical illustration of the proposed growth procedure can be seen in Figure 1.

3.2 Outline of the greedy algorithm

In practice the presented growth procedure consists of two parts: finding an open ear and extending the current 2-connected subgraph, with all the corresponding updates to the BFS tree. The idea of the greedy procedure is to extend S not with the first found open ear but with the best one, based on some heuristic function, from a suitable list of candidates. To implement such an algorithm there are two main parts: defining a heuristic function and maintaining a suitable candidate list.

As the goal of the algorithm is to find a minimal 2-connected dominating set, the heuristic function should be defined in a way to make this possible. A natural

requirement for the heuristic function is to have a preference for small ears that are adjacent to a high number of nodes that are not already dominated or inside the current partial dominating set S. Let us define the $\hat{N}[P]$ as the set of neighboring nodes to P, and $N[P] = \hat{N}[P] \cup P$. Now we can define the heuristic function as

$$dom(P) = \frac{|N[P] \setminus N[S]|}{|In(P)|} \tag{12}$$

In Eq. (12) the notation $In(P)$ is used for the set of inner nodes of P. The notation $|\dots|$ is used for the cardinality of a set. The heuristic function $dom(P)$ is equal to the number of nodes that are in $N[P]$ but not already dominated by S divided by the number of inner nodes of the open ear P.

In the growth procedure presented in Algorithm 1, as soon as a new open ear P is found, the subgraph S is extended accordingly. In practice this is not necessary, but instead we can define a candidate list $C = \{P_1, P_2, \cdots, P_M\}$ to which we add P. When there is a sufficient number M of candidates in C, we can select the best ear P_b based on the heuristic function $dom(P)$, and only then expand S and perform the necessary updates to the BFS tree. In the practical implementation of the algorithm some additional effort is needed to maintain the validity of the open ears in the candidate list and to make the calculations computationally efficient, for which details can be seen in [**14**].

4. LOCAL SEARCHES AND GRASP

In this section, we present two correction procedures for the 2-CDS. The first is based on a simple heuristic approach and the second one uses a MIP formulation to explore a larger neighborhood. Later we give a short outline of a GRASP algorithm for the problem of interest.

4.1 Heuristic local search

The first local search procedure we use is similar to the one used in [**14**]. It uses a simple correction procedure in which we iteratively remove nodes that are not necessary. The term "necessary" is used for a node u for which $S_u = S \setminus \{u\}$ is either not a dominating set of G or the graph induced by S_u is not 2-connected. For all nodes in $u \in S$, we can simply check if the graph induced by S_u is a 2-connected graph by using Tarjan's linear time algorithm [**12**]. Let us define the function $CPH(S, u)$ which removes node u from S if the resulting S_u is a 2-CDS. The pseudocode for the local search $LSH(S)$, based on the correction procedure, can be seen in Algorithm 2, and more detailed description can be found in [**14**].

4.2 Advanced local search

In this section we present a more extensive local search for improving the solutions. Let D be the current best found 2-connected dominating set. Let us

Algorithm 2 Pseudo code for local search using heuristic based correction procedure for the 2-CDS

```
S.Check = S
repeat
    Select random u ∈ S.Check
    if (S_u is 2-CDS for G) then
        S = S_u
        S.Check = S
    end if
    S.Check = S.Check \ {u}
until S.Check = ∅
```

consider a set of nodes $I \subset D$ for which we wish to perform the correction. In relation let us define $R = D \setminus I$. The objective of the proposed correction procedure is to find a set of nodes I' for which the graph induced by $D' = R \cup I'$ is 2-connected and D' is a dominating set for graph G and $|I'| < |I|$. Our aim is to select a subsection I, of the 2-connected dominating set, for which it is to possible to have a MIP formulation in an efficient way.

One way to achieve this is to select I in a way that the graph induced by R is 2-connected. With the intention of having a clear presentation, we start with several definitions. Let N be the set of nodes in V that are not dominated by R. Further, let us define X as the set of nodes in N and ones that are connected to nodes in N using paths consisting only of nodes in $V \setminus R$, having a length less or equal to $|I|$. Let us define the set $B \subset R$ as all the nodes $i \in R$ for which there exist $j \in X$ such that $(i, j) \in E$. Let us define the edge set $\hat{E}$ as the set of all edges of the graph induced by the set $X \cup B$ extended by a set of edges that form a cycle with only nodes in B. Let use define the graph $\hat{G}(X \cup B, \hat{E})$ See Fig. 4 for illustration of the used sets.

The correction procedure needs to provide us with a mechanism for finding I', based on some input I, such that $I' \cup R$ is a 2-CDS for G. To achieve this it is sufficient that I' is a 2-CDS for $\hat{G}$ and $|B| \geq 2$. It can be trivially shown that $I' \cup R$ is a dominating set for G. The restriction that the graph induced by R is 2-connected makes it possible to prove that $I' \cup R$ is also 2-connected. To be more precise, we show that for any node $n \in I'$ there exists an open ear to R that contains it. In the following text we give a short outline of the proof.

Since the graph $\hat{G}$ is 2-connected there must be two vertex-disjoint paths for a node i to some node $b \in B$. If any of this two paths P_1, P_2 contain a node $b' \neq b$, $b' \in B$ an open ear obviously exists. Let us show that the assumption that no such node b' exists results in a contradiction. By definition a path $P_1/\ P_2$ contains a node $n_1/n_2 \notin B$ such that an edge $\{n_1,\ b\}/\ \{n_2,\ b\}$ exists. Since the graph $\hat{G}$ is 2-connected there must exist a path P' connecting n_1 to some node $b_1 \in B$ not containing b'. If this path does not contain any nodes in $P_1\ /\ P_2$ than a open ear containing n can be trivial created. In case this is not true, there exists $l \in P'$,

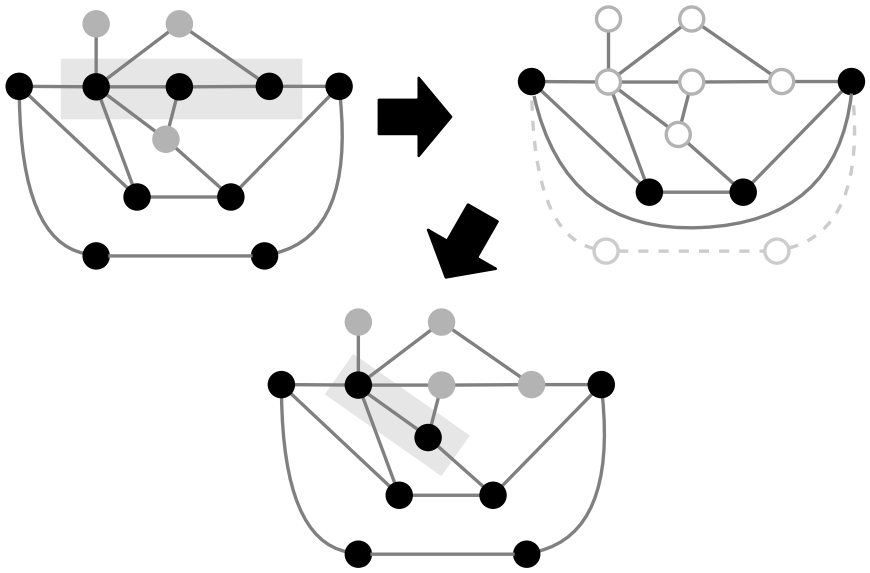

Figure 4: Illustration of node and edge sets used in the problem definition. In Step 1 the black nodes inside the gray rectangle represent the set I used in the correction. In Step 2, white nodes represent the non dominated node set N. The nodes that are considered to be used in the extension of the dominating set are white and represent the set X. The black nodes represent the border nodes B in the dominating set. Edges in the graph used in the subproblem are solid lines. Note that there is an additional edge added to form a cycle with nodes in B. The dashed nodes and edges are not considered in the subproblem. In Step 3, the black nodes in the gray rectangle represent the improved extension I' of the partial dominating set.

$l \in P_2 \cup P_1$ such that there is no other node having the same property after it in P'. Let us assume that $l \in P_1$ than the path P^* consisting of all the nodes in P_1 before l, node l and nodes in P' after l can be combined with P_2 to form an open ear containing n, which is contrary to the initial assumption.

Using this observation we can easily define the MIP formulation for the correction procedure using the MIP formulation given in (1) - (8). A 2-CDS will be found using the mentioned MIP formulation on the new graph $\hat{G}$. With the following additional constraints, which guarantee that all the nodes in B are a part of the 2-CDS.

$$y_i = 1 \qquad i \in B \tag{13}$$

An illustration of this procedure can be seen in Fig. 4. Let us define $CMP(I, D)$ as the results of applying this correction procedure for dominating

set D and test set of nodes I. Using this correction procedure we can define a corresponding local search $LSM(D)$ using the pseudo code given in Alg. 3.

Algorithm 3 Pseudocode for local search using MIP based correction procedure for the 2-CDS

```
S.Check = S
repeat
    Select random u ∈ S.Check
     Find 2-connected S' ⊂ S_u, b_l ≤ |S \ S'| ≤ b_u
    I = S \ S'
    D = CMP(I, S)
    if |D| < |S| then
        S = D
        S.Check = S
    end if
    S.Check = S.Check \ {I}
until S.Check = ∅
```

In it, for a solution S all its nodes should be tested for potential improvement. $S.Check$ will represent the set of all nodes that need to be evaluated. In the main loop, we first select a random node $n \in S.Check$. Based on n, a 2-connected subset $S' \subset S_u$ is selected, and later used to specify the test set $I = S \setminus S'$. The parameter b_u is used to limit the size of the problem that will be solved using MCP. The lower bound b_l is used to define a minimal region that will be evaluated in the correction procedure. In the next step the correction procedure is applied $D = CMP(S, I)$. In case a new best solution D is found the solution S is set to D, and the set of nodes that should be tested is reset. The next step is to remove all the nodes in I from the set of nodes that need to be checked. The main loop is repeated until there are no nodes left to check.

4.3 GRASP

In this subsection we present a short overview for the GRASP method for the 2-CDS based on the presented greedy algorithm. The GRASP algorithm consists of two parts it uses a randomized greedy algorithm to generate initial solutions, and on each one of them a local search procedure is implemented. The randomization can be done straight forward by initially selecting a random root node $r \in G$ for the growth procedure and by defining a new heuristic function $dom_r(P) = \alpha dom(P)$, where α is a random variable from some interval $(1, \alpha_0)$. The pseudo code for the GRASP can be seen in Alg 4.

5. MATHEURISTIC APPROACH

5.1 Efficient exploration of the solution space

Algorithm 4 Pseudo code for GRASP for the 2-CDS

while Not Terminating Conditions **do**
  Generate 2-CDS D based on the randomized greedy algorithm
  $D = LSP(D)$ ▷ Local search procedure
  Check if D is new best solution
end while

The main goal of the proposed algorithm is to efficiently use the proposed MIP formulation as a local correction procedure. In general, the use of a local search or correction procedure is very efficient in the GRASP setting. Before giving details of the method, let us first make a few observations about implementing such a method in case of 2-CDS in a matheuristic setting. Firstly, the use of the CPM explores a larger neighborhood of a solution than CPH, and as a consequence a higher level of improvement can be achieved. The use of a larger neighborhood in CPM, makes it possible to escape some locally optimal solutions in which CPH will be trapped. On the other hand the CPM has a significantly greater computational cost than the CPH. As our previous results have shown, a GRASP algorithm using the heuristic correction procedure can find high quality solutions for the problem of interest.

Although LSM can be directly included in the GRASP algorithm, this may not be the best choice. Our goal is to finding a good balance in the use of the CPH and the CPM. To be more precise, we wish to avoid the "over use" of CPM, in the sense of avoiding its application to solutions that can be improved using CPH. From these observations we can conclude several properties that such a method should posses. As in the case of standard GRASP, the local search should be applied on a large number of solutions. The CPH should be used more frequently than CPM. CPM should be used to help the method escape locally optimal solutions that trap a method based on CPH. The application of CPM should be avoided on low quality solutions. This can be done using a simple concept of only applying the CPM to the best solutions. This concept is best understood through observing the pseudo code of the method given in Alg. 5.

In it firstly, an initial set of N solutions $\mathcal{S}$ will be generated using a GRASP algorithm based on the LSH. Each solution $C \in \mathcal{S}$ will be specified by the set of nodes D in the 2-CDS, and a set of nodes that have not been previously checked using CPM. Let us note that when a set D is added to $\mathcal{S}$ the set of unchecked nodes will be equal to D. In the main loop, from the set $\mathcal{S}$ we select the best solution C for which not all nodes have been check using CPM. Next, a random node $n \in C.Check$ is selected and the corresponding test set I is calculated. Further, $C.Nodes$ is tested for local improvement using $D = CPM(I, C.Nodes)$. The nodes in I are removed from $C.Check$. In case D is a new solutions it is added to $\mathcal{S}$, and the same is done for $LSH(D)$. It is important to note that the new solution will often satisfy $D \neq C$ and $|D| = |C|$. This process is repeated until the stopping criterion is satisfied. In case all potential tests have been performed before that,

Algorithm 5 Pseudocode for a the matheuristic method for the 2-CDS

```
Generate solution set |S| = M using the GRASP(LSH)
repeat
    C = BestNotChecked(S)
    Select random u ∈ C.Check
     I = Candidate(n)
    D = CPM(I, C.Nodes)
    if D ∉ S then
        S = S ∪ D ∪ LSH(D)
    end if
    if S = ∅ then
        Generate new M solutions using GRASP(LSH)
    end if
until Stopping Criterion
```

$\mathcal{S} = \emptyset$, a new group of test solutions will be generated.

5.2 Exploiting common sections of high quality solutions

Experience gained through the generation of solutions should be exploited in solving the problem of interest. In general, GRASP algorithms are capable of efficiently exploring the solutions space. This is achieved through independently generating a large number of solutions. The performance of such algorithms is highly dependent on the quality of the local search and the used greedy algorithm. One of the disadvantages of the GRASP algorithm is the fact that no experience is gathered form previously generated solutions.

In the proposed algorithm, we introduce a new concept for exploiting information gathered in this way. We start from a simple idea that all good solutions are "similar". The idea is that all high quality solutions have a common "core" that is contained in most of them, and likely in the global optimal solution. For instance, in case of the 2-CDS a "core" would be a set of nodes that are contained in all high quality dominating sets. On the other hand, in case a certain number of variables is fixed in a MIP formulation, solving the problem becomes substantially less computationally expensive. Secondly, solutions generated using the proposed algorithm, and population based metaheuristics in general, generate solutions that are to a large extent similar.

We exploit this general idea in case of the 2-CDS for the proposed algorithm. Let use assume that we have generated a set of solutions $\mathcal{S}$, and $\mathcal{B}$ is the set of N best solutions. Let us define the set of common cores for a set of solutions $\mathcal{B}$ in the following way.

$$Core(\mathcal{B}) = \{T \mid (\mathcal{A} \subset \mathcal{B}) \wedge T = \bigcap_{E \in \mathcal{A}} E\} \tag{14}$$

In practice, the set $Core(\mathcal{B})$ represents the set of intersections for each element of

the power set $\mathcal{P}(\mathcal{B})$. In relation we define a set of common cores for a solution C as

$$ComCore(C, \mathcal{B}) = \{T \mid (T = C \cap A) \wedge (A \in Core(\mathcal{B}))\}. \tag{15}$$

For some core C, we can find the optimal solution Opt using the MIP formulation given in(1) - (8), by adding the following additional constraints

$$y_i = 1 \qquad i \in C. \tag{16}$$

There are two important aspects of solving this problem. Firstly, it is likely that it will produce a high quality solution for the problem being solved. Secondly, it gives us information which nodes should be evaluated using the correction function CPM for some solutions D and solved core C. To be more precise, if $C \subset D$, an improvement to the optimal solution containing C can be achieved using $CPM(I, D)$, if $I \cap C \neq \emptyset$. As a consequence, only nodes $n \in C$ should be used in generating the test sets I. Let us not that in case we have solved the MIP for a set cores C_1, C_2,.., C_n, then we only need to check nodes n that are elements of $\bigcap_{i=1..n} C_i$.

5.3 Implementation details

In this section we give a full description of the proposed algorithm and give some implementation details. The pseudo code for the proposed method can be seen in Algorithm 6.

In practice, it is the same as the algorithm presented in the previous section except that it also incorporates the use of common cores. To be more precise, for a current solution C, in the main loop, it checks if all the nodes in C need to be tested. In case this is true, it selects a random common core $Core$ form $ComCore(C.Nodes, \mathcal{B})$ satisfying size constraints. Solves the corresponding restricted MIP, and adds the new solutions D to the solution set $\mathcal{S}$. Further, it adds the newly tested core to the set of cores. Next, for every test solution T the set $T.Check$ is updated based on the new core $Core$.

In the implementation of the method, when solving the subproblem $CMP(I, C.Nodes)$ it is not necessary to solve the MIP formulation to optimality since it is just a part of the heuristic algorithm. In practice, our goal is to avoid making the CMP overly computationally expensive because of which a time limit T_{CMP} is given for solving the subproblem. For the same reason a time limit T_{Core} is used for the subproblem related to common cores. It is important to note that if the subproblem $MIP(Core)$ is not solved to optimality, than the $Core$ should not be added to the set $Cores$, which is used to limit the nodes that should be tested for potential improvement.

As in the case of CMP, bounds L_{Core}, U_{Core} are introduced on the size of the subproblem that will be solved for a core. In the implementation, we do not generate the complete set of common cores for a solution C and select a random one. Instead, we use a randomized algorithm to construct a single core satisfying the size constraints.

Algorithm 6 Pseudo code for a the matheuristic method for the 2-CDS which exploits common cores

Generate solution set $\mathcal{S}$ using the GRASP((LSH)
repeat
  $C = BestNotChecked(\mathcal{S})$
  if $C.Nodes == C.Check$ **then**
    Select a random $L_{Core} < |Core| < U_{Core}$ from $ComCore(C.Nodes, \mathcal{B})$
    $D = MIP(Core)$
    if $D \notin \mathcal{S}$ **then**
      $\mathcal{S} = \mathcal{S} \cup D \cup LSH(D)$
    end if
    $Cores = Cores \cup Core$
    $FixCandidates(\mathcal{S}, Cores)$
  end if
  Select random $u \in C.Check$
  $I = Candidate(n)$
  $D = CMP(I, C.Nodes)$
  if $D \notin \mathcal{S}$ **then**
    $\mathcal{S} = \mathcal{S} \cup D \cup LSH(D)$
  end if
  if $\mathcal{S} = \emptyset$ **then**
    Generate new N solutions using GRASP(LSH)
  end if
until Stopping Criterion

The MIP for the subproblems is solved using lazy cuts. More precisely, the constraints given in(5), (6) are added in this way. Our initial empirical test have shown that it is more efficient to add a single cut at each step, similar to the results in [**4**]. The 2-connectivity has been tested using Tarjan's linear time algorithm [**12**].

6. RESULTS

In this section, we present the results of the performed computational experiments used to evaluate the proposed matheuristic (MH) approach for solving the 2-CDS. The tests are divided into two groups. In the first, we compare the proposed method to the basic $GRASP$ algorithm and MIP formulation from [**4**] (Bu) and [**20**] (For) on general graphs of small to medium size. In the second group of test, we compare the proposed method to the basic GRASP algorithm on medium and large Unit Disc Graphs (UDGs). Both algorithms have been implemented in C# using Microsoft Visual Studio 2015. In case of the matheuristic approach the mathematical model has been solved using CPLEX Version 12.6.1. The integration of the solver to the project has been done using the Concert Technology in the .NET framework. 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. The parameters for specifying MH and $GRASP$ were the following: the maximal size of the candidate list is $M = 500$; the randomization parameter is $\alpha_0 = 1.25$. The value of M influenced the time needed to generate a solution, while the value of α_0 influences the quality of such solutions. The limit on the size of the subproblem in case of CMP, corresponding to the number of removed nodes I, was based on the solution being improved S and had the values $b_l = 2$ and $b_u = min(0.5|S|, 10)$ The bounds for the size of a common core for a solution S, were $L_{Core} = 0.5|S|$ and $U_{Core} = 0.75|S|$. These values have been chosen empirically, in a way that the related subproblems could be solved in a short time. The time limit for solving CMP and $MIP(Core)$ were 0.5 and 1 second, respectively. Let us note that these values have been rarely reached in the execution of the algorithm.

6.1 General graphs

In this subsection, we present a comparison of the proposed method MH to the basic GRASP algorithm and the MIP based methods from [**4**] and [**20**]. The comparison has been made on 41 random graphs having 30 - 200 nodes and edge densities between 5-70%. The graph instances are the same as in [**4**], which have also been used in [**3**]. In the Table 1, we observe the computational times needed to solve the MIP formulation or find the optimal solutions in case of $GRASP$ and MH. In all the runs for MH and For, a time limit of 3600 seconds was given. For Bu and $GRASP$ the values have been taken from corresponding articles. In case of For, we have implemented the method from [**20**]. The same implementation is used for the calculation of subproblems occurring in the MH method. Due to the probabilistic nature of MH, for each test instance 10 separate runs, with different random seeds, have been performed. In Table 1, we show the average and worst execution times and the related standard deviation.

From these results, we can first see that the Bu formulation has significant advantages over For. In case of highly dense graphs the MH and $GRASP$ have a very good performance and manage to find optimal solutions in a significantly lower time than the MIP based methods. Only MH and Bu managed to find optimal solutions for all the test instances. The advantage of using the MH approach is most evident in case of the largest problem instances. In case of problem instances having 200 nodes in only one case the average execution time is worse than Bu. When we compare the total calculation time, MH needed 321.74 seconds to find all the optimal solutions on average, while in case of Bu it was 799 seconds. The total execution time for the worst runs of MH is 934 seconds which still makes it competitive with Bu. Let us note that in case of MH this is the time needed to find the optimal solution, while Bu also confirms that a solution is optimal. The computational time is lower than For for an order of magnitude. It is important to note that the execution time of MH, would significantly vary, in case of some instance the worst execution time would be even 5 times higher than the average. Let us note that it is expected that the performance of the proposed algorithm

could be significantly improved if the formulation from [**4**] was used instead of the one from [**20**].

6.2 Unit disc graphs

As previously stated one of the most important applications of the DSP is in the field of wireless and ad-hoc networks. Such systems are commonly represented using unit disc graphs. Due to this fact, we have also conducted computational experiments on this type of graphs.

The proposed method is evaluated on a wide range of unit disc graphs having between 100 to 1000 nodes, with different edge densities d. The method for generating problem instances is the following. In a box having dimensions $[0,1] \times [0,1]$ αn random points are uniformly distributed, where $\alpha = 1.1$. These points correspond to potential graph nodes. The half diameter of the unit discs is calculated using the following formula.

$$r = \frac{d}{\pi\sqrt{n}} \tag{17}$$

The used density values are 2, 3, 4 and 5. An edge would exist in the graph if the distance between two nodes was less than $2r$. In practice this means that the total area of the unit discs is 2, 3, 4 or 5 times the area of the box. If the generated graph had a 2-connected subgraph containing n nodes, the subgraph is accepted as a problem instance. For each combination of n(number of nodes) and d(density), five problem instances were generated using different random seeds. In total 100 different test instances are generated, and can be found at [**13**].

We compare the original GRASP algorithm from [**14**] with MH. In case of problems on medium and large UDGs the MIP formulation from [**20**] had a very poor performance and could not find optimal solutions even in the case of graphs having 100 nodes in reasonable time. From our observations, we assume that this issue comes from the fact that a much higher number of lazy cuts needs to be added, and as a consequence the computational cost grows dramatically. Because of this, the MH approach did not exploit the concept of common cores of high quality solutions.

The comparison of MH and $GRASP$ has been done on each pair (n, d). For both methods a fixed time limit, dependent on the problem size, is given. We have compared he following values: the average size of the solution and the number of instances a method had found a unique best solution. We also present the minimal and maximal improvement MH had achieved when compared to GRASP. The results of the conducted computational experiments can be seen in Table 2.

From the results presented in this table, we first notice that the methods achieve the same quality of solutions in case of problems having 100 nodes. We expect that this is due to the fact that both methods manage to find optimal solutions. Overall MH, had a significantly better performance having a better/worse average quality of solutions for 14/1 out of 20 pairs (n, d) than $GRASP$. $GRASP$

managed to find better/worse solution for 4/66 instances than MH. In case of the largest problem instances having 750 and 1000 nodes MH managed to achieve an improvement for all test instances. The average improvement of MH overall was around 4%, while it was close to 10% in case of largest problem sizes.

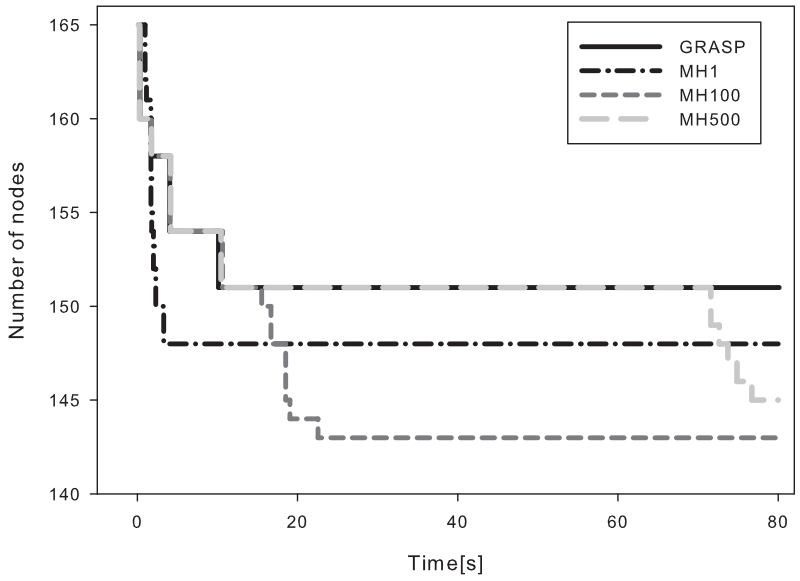

Figure 5: Computational time for intermediate solutions for one problem instance having 750 nodes and density $d = 3$.

In our second group of tests we observe the speed of convergence of the MH and $GRASP$. The behavior of MH is similar for different graph sizes, which we illustrate on one representative problem instance given in Figure 5. In case of MH, we show that the behavior is dependent on the size of the initial population $M = 1, 100, 500$. In the further text we will use the notation $MH\{k\}$ for the MH method using a population size of k. In case of $MH1$, the method could be understood as a GRASP algorithm that uses a MIP local search. Let us note, that the GRASP algorithm based on a local search presented in Subsection had a notable worse performance, so we did not include it in the evaluation.

In the general case the use of $GRASP$, and consequently MIP with a larger value of M, has an advantage in case of finding high quality solutions in very short times. $MH1$ manages to find medium quality solutions the fastest, in case the constructive algorithm and local search are relatively expensive. In case of long execution times $MH1$ is not competitive with $MH100$ and $MH500$. This is due to the fact that it attempts improving random, often low quality, solutions at a high

cost. In case of larger initial populations, this problem is avoided since a only high quality solutions are tested. It is important to note that these solutions are often in different areas of the solution space. On the other hand, if the initial size of the population is too large too much time is spent in generating initial solutions. From these observations we expect that it is better to use a stagnation criterion instead of a fixed size of the initial population.

From observing the behavior of the algorithm, we have noticed that the CPM has often been applied on the same subproblems. This was significantly more evident in case of UDGs than in case of general graphs. In our implementation we did not include any tracking of solved subproblems, but we believe that this could improve the performance of the algorithm. The second important observation about the execution of the proposed algorithm, is that in case of very long executions the method would explore lower quality solutions. Their simple exclusion often resulted in the search being trapped in local optimal solutions. Exploring efficient methods for excluding such solutions could also be a promising avenue of improving the performance of the method.

7. CONCLUSIONS

In this paper we have presented a matheuristic approach for solving the 2-CDS. The method is designed for finding high quality solutions for large graphs. It extends a GRASP algorithm with a MIP based local search. The focus of the paper is on efficiently combining a low computational cost heuristic correction procedure, which explores small neighborhoods of a solution, and a high cost MIP based one that explores large neighborhoods.

One of the main advantages of the approach is that the subproblem is also the 2-CDS. To be more precise, the 2-CDS is either solved on a smaller graph or the same size with fixed values for a set of variables in the MIP formulation. Our computational experiments have shown that the method is competitive with the state-of-the-art. Further, it has been shown that the proposed method has a significantly better performance than the MIP formulation used for solving the subproblem. On the other hand, alternative MIP formulations for the 2-CDS can be easily included, and it is expected that their performance can be improved in a similar way.

In the future, we plan to explore the application of this type of approach to other graph problems like the general k-m-CDS, other variations of the DSP and the vehicle routing problem. Another promising direction for future research is finding efficient methods for excluding areas on individual solutions on which the local search should be applied.

REFERENCES

1. A. Ahangar, M. Henning, V. Samodivkin, I. Yero *Total Roman domination in graphs Appl. Anal. Discrete Math.* , 10(1):501 – 517, 2016.
2. N. Ahn and S. Park. *An optimization algorithm for the minimum k-connected m-dominating set problem in wireless sensor networks. Wirel. Netw.* , 21(3):783–792, 2015.
3. A. Buchanan, J. S. Sung, V. Boginski, and S. Butenko. *On connected dominating sets of restricted diameter. Eur. J. Oper. Res.*, 236(2):410 – 418, 2014.
4. A. Buchanan, J. S. Sung, S. Butenko, and E. L. Pasiliao. *An integer programming approach for fault-tolerant connected dominating sets. INFORMS J. Comput.*, 27(1):178–188, 2015.
5. M. Caserta and S. Voss. *A math-heuristic Dantzig-Wolfe algorithm for capacitated lot sizing. Ann. Math. Artif. Intel.*, 69(2):207–224, 2013.
6. J. Cáceres, C. Hernando, M. Mora, I. Pelayo and M. Puertas *Perfect and Quasiperfect domiantion in trees Appl. Anal. Discrete Math.* , 10(1):46 – 64, 2016.
7. F. Dai and J. Wu. *On constructing k-connected k-dominating set in wireless ad hoc and sensor networks. J Parallel. Distr. Com.*, 66(7):947–958, 2006.
8. D.-Z. Du and P.-J. Wan. *Connected dominating set: theory and applications.* Springer Science & Business Media, 2012.
9. B. Gendron, A. Lucena, A. S. da Cunha, and L. Simonetti. *Benders decomposition, branch-and-cut, and hybrid algorithms for the minimum connected dominating set problem. INFORMS J. Comput.* , 26(4):645–657, 2014.
10. A.-R. Hedar and R. Ismail. *Hybrid genetic algorithm for minimum dominating set problem.* In D. Taniar, O. Gervasi, B. Murgante, E. Pardede, and B. O. Apduhan, editors, *Computational Science and Its Applications – ICCSA 2010: International Conference, Fukuoka, Japan, March 23-26, 2010, Proceedings, Part IV*, pages 457–467. Springer, Berlin, Heidelberg, 2010.
11. C. K. Ho, Y. P. Singh, and H. T. Ewe. *An enhanced ant colony optimization metaheuristic for the minimum dominating set problem. Appl. Artif. Intell.* , 20(10):881–903, 2006.
12. J. Hopcroft and R. Tarjan. *Algorithm 447: efficient algorithms for graph manipulation. Commun. ACM* , 16(6):372–378, 1973.
13. R. Jovanovic. *Benchmark data sets for the 2-cds: Unit disc graphs, 2016.*
14. R. Jovanovic, I. S. Bayram, and S. Voss. *A GRASP approach for solving the 2-connected m-dominating set problem. arXiv preprint arXiv:1609.05662*, 2016.
15. R. Jovanovic, T. Nishi, and S. Voss. *A heuristic approach for dividing graphs into bi-connected components with a size constraint. J Heuristics*, 23(2):111–136, Jun 2017.
16. R. Jovanovic and M. Tuba. *Ant colony optimization algorithm with pheromone correction strategy for the minimum connected dominating set problem. Comput. Sci. Inf. System.*, 10(1):133–149, 2013.

17. R. Jovanovic, M. Tuba, and D. Simian. *Ant colony optimization applied to minimum weight dominating set problem.* In *Proceedings of the 12th WSEAS international conference on Automatic control, modelling & simulation*, pages 322–326. World Scientific and Engineering Academy and Society (WSEAS), 2010.

18. C. Lagos, G. Guerrero, E. Cabrera, S. Niklander, F. Johnson, F. Paredes, and J. Vega. *A matheuristic approach combining local search and mathematical programming. Sci. Programming.*, 2016. Article ID 1506084, 7 pages, doi:10.1155/2016/1506084.

19. E. A. Lalla-Ruiz and S. Voss. *Towards a matheuristic approach for the berth allocation problem.* In M. P. Pardalos, G. M. Resende, C. Vogiatzis, and L. J. Walteros, editors, *Learning and Intelligent Optimization: 8th International Conference, Lion 8, Gainesville, FL, USA, February 16-21, 2014. Revised Selected Papers*, pages 218–222. Springer, Cham, 2014.

20. V. Leal do Forte, A. Lucena, and N. Maculan. *Formulations for the minimum 2-connected dominating set problem. Electronic Notes in Discrete Mathematics*, 41:415 – 422, 2013.

21. Y. Li, Y. Wu, C. Ai, and R. Beyah. *On the construction of k-connected m-dominating sets in wireless networks. J Comb. Optim.*, 23(1):118–139, 2012.

22. M. Morgan and V. Grout. *Finding optimal solutions to backbone minimisation problems using mixed integer programming.* In *Proceedings of the 7th International Network Conference (INC 2008)*, pages 53–64, 2008.

23. H. E. Robbins. *A theorem on graphs, with an application to a problem of traffic control. Am. Math. Mon.* , 46(5):281–283, 1939.

24. E.-G. Talbi. *Combining metaheuristics with mathematical programming, constraint programming and machine learning. 4OR*, 11(2):101–150, 2013.

25. M. T. Thai, N. Zhang, R. Tiwari, and X. Xu. *On approximation algorithms of k-connected m-dominating sets in disk graphs. Theor. Comput. Sci.*, 385(1–3):49 – 59, 2007.

26. F. Wang, M. T. Thai, and D. Z. Du. *On the construction of 2-connected virtual backbone in wireless networks. IEEE T Wirel. Commun.* , 8(3):1230–1237, March 2009.

27. W. Wang, D. Kim, M. K. An, W. Gao, X. Li, Z. Zhang, and W. Wu. *On construction of quality fault-tolerant virtual backbone in wireless networks. IEEE/ACM T on Netw.*, 21(5):1499–1510, 2013.

28. J. Yu, N. Wang, G. Wang, and D. Yu. *Connected dominating sets in wireless ad hoc and sensor networks — a comprehensive survey. Comput. Commun.* , 36(2):121 – 134, 2013.

Raka Jovanovic (Received 27.02.2019) Qatar Environment and Energy (Revised 25.08.2019)
Research Institute (QEERI),
Hamad bin Khalifa University,
PO Box 5825, Doha,
Qatar
E-mail: *rakabog@yahoo.com*

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

Table 1: Comparison of execution times for $GRASP$, MH, For and Bu on general graphs for finding optimal solutions. The symbol $*$ has been used in case the method did not find an optimal solution. *Opt* correspond to the size of the optimal solution.

Instance	Opt	Bu	For	$GRASP$	MH	
					Avg(Std.)	Max
v30_d10	18	0.02	0.01	<0.01	0.01(0.01)	0.03
v30_d10	18	0.02	0.01	<0.01	0.01(0.01)	0.03
v30_d20	8	0.04	0.08	0.03	0.03(0.03)	0.08
v30_d30	5	0.02	0.03	0.03	<0.01	0.01
v30_d50	3	0.01	0.02	<0.01	<0.01	<0.01
v30_d70	3	0.05	0.36	<0.01	<0.01	<0.01
v50_d10	14	0.05	0.09	0.06	0.01(0.01)	0.02
v50_d20	7	0.04	0.12	<0.01	0.05(0.03)	0.09
v50_d30	5	0.09	0.50	0.02	0.02(0.02)	0.06
v50_d50	3	0.08	0.19	0.03	0.02(0.01)	0.04
v50_d70	3	0.08	0.41	<0.01	<0.01	<0.01
v70_d5	34	0.01	0.01	13.00	1.18(0.45)	1.30
v70_d10	14	0.10	0.30	2.90	1.33(0.89)	2.62
v70_d20	8	0.13	0.93	0.04	0.05(0.05)	0.14
v70_d30	5	0.21	0.59	0.08	0.06(0.08)	0.26
v70_d50	3	0.08	0.56	0.10	0.02(0.02)	0.05
v70_d70	3	0.16	0.33	<0.01	<0.01	<0.01
v100_d5	28	0.20	0.84	17.00	1.76(0.94)	2.45
v100_d10	14	0.29	1.02	6.40	2.23(1.92)	6.24
v100_d20	8	0.73	1.61	2.80	1.60(0.99)	3.09
v100_d30	6	1.00	2.84	0.17	0.07(0.06)	0.19
v100_d50	4	0.53	1.82	0.03	0.01(0.01)	0.02
v100_d70	3	1.34	0.79	<0.01	<0.01	0.01
v120_d5	27	0.17	0.83	2.00	4.59(2.67)	8.58
v120_d10	14	0.60	1.82	30.00	8.51(8.84)	28.92
v120_d20	8	3.83	39.30	74.00	6.83(4.06)	13.10
v120_d30	6	1.22	4.02	1.40	0.78(0.89)	2.80
v120_d50	4	3.29	4.23	0.01	0.02(0.02)	0.05
v120_d70	3	0.52	1.90	0.01	<0.01	0.01
v150_d5	28	0.93	25.47	*	13.59(10.64)	30.96
v150_d10	15	3.94	33.17	32.00	4.25(2.73)	10.28
v150_d20	9	6.56	82.19	1.00	3.49(3.13)	9.93
v150_d30	6	2.64	21.81	14.00	2.96(1.92)	5.00
v150_d50	4	3.10	9.44	0.02	0.08(0.06)	0.17
v150_d70	3	4.81	2.63	0.09	0.01(0.01)	0.02
v200_d5	29	27.47	859.12	*	74.24(77.14)	225.85
v200_d10	16	172.73	*	*	126.58(100.03)	246.56
v200_d20	9	386.55	2046.76	56.00	65.43(103.55)	326.96
v200_d30	7	157.67	3580.00	0.27	0.37(0.32)	1.09
v200_d50	4	8.42	37.01	2.63	1.53(2.27)	6.76
v200_d70	3	9.48	12.81	0.02	0.02(0.01)	0.03

Table 2: Comparison of GRASP and the matheuristic approach (MH) in case of UDGs

Nodes	Time[s]	Avg. Size		Num. Imp.		MH Imp.	
		GRASP	MH	GRASP	MH	Max	Min
Density $d = 2$							
100	15	41.2	41.2	0	0	0	0
200	30	78.4	78.4	1	1	1	-1
500	60	206.0	203.4	0	5	5	1
750	120	297.4	292.4	0	5	8	1
1000	300	389.6	379.0	0	5	13	8
Density $d = 3$							
100	15	21.0	21.0	0	0	0	0
200	30	37.6	36.8	0	4	1	0
500	60	95.8	91.2	0	5	9	2
750	120	145.0	134.0	0	5	16	6
1000	300	196.2	179.2	0	5	20	14
Density $d = 4$							
100	15	10.8	10.8	0	0	0	0
200	30	20.6	20.4	0	1	1	0
500	60	54.2	52.8	1	3	3	-1
750	120	84.2	78.6	0	5	8	3
1000	300	114.4	103.8	0	5	13	9
Density $d = 5$							
100	15	7.2	7.2	0	0	0	0
200	30	12.8	13.0	1	0	0	-1
500	60	35.6	34.6	1	3	3	-1
750	120	54.2	51.2	0	5	5	2
1000	300	74.8	67.6	0	5	10	4
Average		98.85	94.83	0.2	3.2	5.8	2.3