Optim Lett (2017) 11:1155–1167
DOI 10.1007/s11590-016-1017-5

ORIGINAL PAPER

A metaheuristic approach to the dominating tree problem

Zorica Dražić[1] · **Mirjana Čangalović**[2] · **Vera Kovačević-Vujčić**[2]

Received: 31 January 2014 / Accepted: 3 February 2016 / Published online: 29 February 2016

Abstract This paper considers a recently introduced NP-hard problem on graphs, called the dominating tree problem. In order to solve this problem, we develop a variable neighborhood search (VNS) based heuristic. Feasible solutions are obtained by using the set of vertex permutations that allow us to implement standard neighborhood structures and the appropriate local search procedure. Computational experiments include two classes of randomly generated test instances and benchmark test instances from the literature. Optimality of VNS solutions on small size instances is verified with CPLEX.

Keywords Dominating tree problem · Graphs · Variable neighborhood search · Optimization

1 Introduction

The dominating tree problem, discussed in this paper, has been recently introduced by Shin et al. in [13]. This problem is defined as follows: Let $G = (V, E)$ be an undirected, connected, edge-weighted graph, where V denotes the set of vertices and E denotes the set of edges. To each edge $e \in E$, a non-negative weight w_e is assigned.

✉ Mirjana Čangalović
canga@fon.bg.ac.rs

Zorica Dražić
zdrazic@matf.bg.ac.rs

Vera Kovačević-Vujčić
verakov@fon.bg.ac.rs

1 Faculty of Mathematics, University of Belgrade, Studentski Trg 16, Belgrade, Serbia

2 Faculty of Organizational Sciences, University of Belgrade, Jove Ilića 154, Belgrade, Serbia

A tree $T = (V(T), E(T))$ of graph G is called *dominating* if each vertex $v \in V$ that is not in T is adjacent to a vertex in T. The weight of tree T is defined as $\sum_{e \in E(T)} w_e$, i.e. as the sum of all edge weights in T. Now, the dominating tree problem (DTP) is to construct a dominating tree T of graph G with the minimal weight.

The DTP has several applications in network design and network routing. Multicasting is one example, given in [13], whose goal is simultaneous delivery of the same data to a group of destination computers. Servers are connected by a tree network structure T, and all other computers are one hop away from a server. If weights represent the cost, or energy to transmit data from one server to another, the sum of edge weights in T equals to overall cost to transmit data from one server to all others.

In [13] the DTP is proved to be NP-hard in the general case, and an approximation framework is provided. Due to the high runtime complexity of this approximation algorithm, the authors propose a polynomial time heuristic. They also propose an integer programming formulation of the DTP.

The first metaheuristic approaches for solving the DTP were proposed in [14], where the authors implemented two swarm inteligence techniques: artificial bee colony and ant colony optimization. The ant colony optimization algorithm produced better results on most of large instances, but it was slower than the artificial bee colony optimization algorithm.

Problems related to the DTP are: *the connected dominating set problem* (CDSP) [5,11,15,16] and *the tree cover problem* (TCP) [1,3,4]. Subset D of vertices of graph G is a *dominating set* of G if each vertex of G is either in D or adjacent to a vertex in D. Now, the CDSP can be formulated as follows: For a given graph, find a minimum size connected dominating set. In some variants of this problem, vertices might have weights and the problem is formulated as to minimize the total weighted sum of the vertices that form a connected dominating set. Note that in the DTP weights are associated with edges and not with vertices.

On the other hand, the TCP considers a graph where each edge has a nonnegative weight. The problem is to find a tree T of graph G with the minimal weight that represents a vertex cover of G, i.e. every edge of G has at least one endpoint in T. In this case, the resulting tree represents an edge dominating set, contrary to the DTP, where the resulting tree is a vertex dominating set.

2 Variable neighborhood search for DTP

The variable neighborhood search (VNS) is a local search based metaheuristic proposed by Mladenović and Hansen [8] in 1997. The concept of the basic VNS can be outlined as follows: The main idea is to use more than one neighborhood structure and to proceed with their systematic change in search for a better solution. Given an incumbent solution, the shaking procedure generates randomly a feasible solution in the current neighborhood. Then, a local search is applied around the generated feasible solution in order to obtain a possibly better solution than the incumbent. If the local search gives a better solution, it becomes the new incumbent. Otherwise, the neighborhood is changed.

A detailed description of different VNS variants can be found in [6,7]. An extensive computational experience with various optimization problems shows that the VNS often gives high-quality solutions in a reasonable time. In particular, we have shown that the VNS approach outperforms genetic algorithms in case of some NP-hard graph optimization problems [9,10]. This experience motivated us to apply the VNS approach to the recently introduced dominating tree problem and to compare it with the existing swarm intelligence-based approaches: ant and bee colony optimization algorithms.

The main characteristics of the VNS metaheuristic applied to the DTP are the following. The feasible solution set X contains all permutations of vertex indices from graph G. To each permutation $x \in X$, the corresponding dominating tree T of G is assigned and the objective function value of x is obtained as the sum of all edge weights from T. The neighborhood structures and the shaking step are defined in the usual way for searching in a set of permutations. The details of the VNS for the DTP are given in a pseudo-code in Fig. 1 and explained in more details below.

The feasible solution set To each vertex $v \in V$ of graph G, let us assign an unique integer index number. The set of feasible solutions X is defined as the set of all permutations of vertex indices from graph G. To each permutation $x \in X$, the corresponding dominating tree T of G is assigned by the procedure *DominatingTree* described in Fig. 2. The procedure first generates a dominating set B of vertices in graph G by including vertices one by one from the beginning of permutation x, until this set becomes a dominating set of G. If subgraph G_B of G induced by B is not connected, the procedure adds more vertices to set B using the order given by x, until G_B becomes connected. The procedure then finds MST(G_B) as the minimal spanning three T of G_B, which represents a dominating tree of G, since it contains all vertices from dominating set B of G, and G_B is connected.

Select a set of neighborhood structures N_k, $k = k_{\min}, \ldots, k_{\max}$
that will be used in the search;
$T \leftarrow$*MST (G)*;
$T \leftarrow$*RemoveLeaves(T)*;
$x \leftarrow$ *MakePermutation(T)*;
$f \leftarrow \sum_{e \in E(T)} w_e$
$x^* \leftarrow x$, $f^* \leftarrow f$;
repeat the following steps **until** the stopping criterion is met
    $k \leftarrow k_{\min}$;
1: **repeat** the following steps **until** $k > k_{\max}$
        $x' \leftarrow$ *Shaking*(x, k);
        $x'' \leftarrow x'$
        $f'' \leftarrow$ *LocalSearch*(x'');
        if $f'' < f^*$ **then**
            $x^* \leftarrow x''$, $f^* \leftarrow f''$ and **goto** 1;
        elseif $f'' == f^*$
            With probability p set $x^* \leftarrow x''$, $f^* \leftarrow f''$ and **goto** 1;
        endif
        $k \leftarrow k + 1$;
    end
end
Stop. Point x^* is an approximative solution of the problem.

Fig. 1 The VNS algorithm for DTP

Fig. 2 Procedure *DominatingTree*

```
Input: Permutation x of vertices
Output: The corresponding dominating tree T
B ← ∅ ;
i ← 0;
repeat the following steps until B is dominating set of G
        B ← B ∪ {x[i]} ;
        i ← i + 1;
end
while G_B is not connected graph do
        B ← B ∪ {x[i]} ;
        i ← i + 1;
end
T ← MST(G_B);
```

The objective function value The objective function value $f(x)$ for a feasible solution $x \in X$ is calculated as the sum of all edge weights of the corresponding dominating tree T, obtained by procedure *DominatingTree*.

The neighborhood structures For $k \geq 2$ we define neighborhood $N_k(x)$ of $x \in X$ as a set of all permutations which differ from x in no more than k positions. During the search, the VNS uses neighborhood structures $N_k, k = k_{\min}, \ldots, k_{\max}$, where $k_{\min}$ and $k_{\max}$ are given parameters.

Shaking step The shaking procedure chooses randomly a solution x' in neighborhood $N_k(x)$ as follows. First, we choose k random numbers from $\{1, \ldots, |V|\}$, representing the positions in permutation x. Next, we permute at random the elements of x from these positions. The resulting vector is denoted by x'.

The local search The local search is defined by procedure *LocalSearch*, given in Fig. 3. Starting from solution x', obtained by shaking, the procedure explores a small neighborhood of x', searching for a solution with smaller objective function value. The neighborhood which is explored consists of all solutions obtained from x' by swapping two of its elements, one from the corresponding dominating set B and the other from $V \setminus B$. The first improvement strategy is used, i.e. as soon as the local search finds a solution with smaller objective function value, the search is continued from this solution. Scanning of the neighborhood is performed as follows: First, $|V| - |B|$ neighbors are examined by swapping the first element of x' with elements of $V \setminus B$, with indices $|V|, |V|-1, \ldots, |B|+1$, respectively. If better solution is not found, we perform the same swapping procedure with elements of x' having indices $2, 3, \ldots, |B|$. The local search stops when the whole neighborhood of the current solution is searched and no further improvement can be made. The best found solution is denoted as x''.

After obtaining the solution x'' by the local search, we have to compare it with the incumbent solution x in order to make a decision wether to accept it or not. In the basic VNS, a move to the new solution x'' is made only if $f(x'') < f(x)$, i.e. the objective function value of solution x'' is smaller than the objective function value of solution x. Considering the permutation-based representation described above, there are different feasible permutations yielding the same objective function value. Moving from one to another such solution, we may diversify the search and explore different regions, increasing in this way the chance of finding a better solution. However, if we do this move every time, we could get trapped in a cycle. To avoid this problem, we use

```
Input: Permutation x'' of vertices
Output: x'', f''
T'' ← DominatingTree(x'');
f'' ← Σ_{e∈E(T'')} w_e;
impr ← true;
2: repeat the following steps until not impr
    impr ← false;
    for i ← 0 to |T''| − 1 do
        for j ← |V| − 1 downto |T''| do
            xtmp ← x'';
            xtmp[i] ← x''[j];
            xtmp[j] ← x''[i];
            Ttmp ← DominatingTree(xtmp);
            ftmp ← Σ_{e∈E(Ttmp)} w_e;
            if ftmp < f'' then
                f'' ← ftmp, x'' ← xtmp, T'' ← Ttmp, impr ← true and goto 2;
            endif
        end
    end
end
```

Fig. 3 Procedure *LocalSearch*

parameter p which represents the probability of moving from one to another solution with the same objective function value. Considering this, after the local search, we have three possibilities:

- If $f(x'') < f(x)$, we move to x'' and continue the search with the same neighborhood N_k from x''.
- If $f(x'') > f(x)$, we continue the search with the same x and the next neighborhood $N_{k+1}(x)$.
- If $f(x'') = f(x)$, we move to x'' with probability p and continue the search from x'' with the same neighborhood and with probability $1 - p$ we continue the search with the same x and the next neighborhood $N_{k+1}(x)$.

The stopping criterion Whenever k_{max} is attained, the search continues with the first neighborhood $N_{k_{\min}}$. This is repeated until some stopping criterion is met. Possible stopping conditions can be the maximum CPU time allowed, the maximum number of iterations, the maximum number of iterations between two improvements, etc. In this VNS implementation, we use the following combination: the algorithm stops when either the maximum number of iterations $iter_{\max}$ or the maximum CPU time $t_{\max}$ is exceeded.

The initial solution The initial permutation can be generated randomly or by using special heuristics. In our implementation, we use the following heuristic approach. First, we construct a minimal spanning tree of G (MST(G)) and then remove all its leaves in order to obtain a tree T. Since all vertices of G are either in T or adjacent to some vertex in T, T is a dominating tree for G. Now, the initial permutation x is taken to be a permutation containing the indices of vertices from T followed by indices of the remaining vertices of G. The objective function value is equal to the sum of all edge weights in T. The described heuristic is implemented in the first three lines of the pseudo-code in Fig. 1.

During the whole VNS procedure, the minimal spanning tree problem is solved by the well known Prim's algorithm [12].

Input parameters for the VNS are the minimum and the maximum number of neighborhoods that should be searched, $k_{\min}$ and $k_{\max}$, the maximum number of iterations $iter_{\max}$, the maximum CPU time allowed $t_{\max}$, and the probability p of moving from one solution to another with the same objective function value.

3 Experimental results

In this section, we present experimental results obtained by the proposed VNS algorithm for solving the DTP. The algorithm was implemented in C programming language. All computational experiments were performed on Intel Core I7-4702MQ 2.2 GHz with 4 GB RAM under Windows XP operating system.

The first group of experiments was performed in order to adjust the key VNS parameters and to analyze their influence on the VNS algorithm for the DTP. As it is well known, the most important parameters in VNS implementations are the values of $k_{\min}$ and $k_{\max}$, which determine the number of different neighborhood structures used during the search process. In our VNS implementation, value $k_{\min} = 2$ is a natural lower bound for k because neighborhood $N_k(x)$ of $x \in X$ is defined as the set of all permutations of vertex indices that differ from x in no more than k positions, which implies $k \geq 2$. In order to find the most suitable values of $k_{\max}$ and probability p for the VNS approach to the dominating tree problem, we have performed experiments with different values of $k_{\max}$ and p on the set of benchmark instances from literature [14]. For each value of $|V| \in \{50, 100, 200, 300, 400, 500\}$ there are three different test instances. The VNS has been run 20 times for each instance with the following stopping criterion: the algorithm stops when either the maximum number of iterations $iter_{\max} = 1000$ or the maximum CPU time $t_{\max} = 600s$ is exceeded.

The results of experiments are summarized in Table 1, organized as follows: In the first column $Inst$ the instance name is given, containing the information about the number of nodes. For each $p \in \{0, 0.1, 0.2, 0.5\}$ and $k_{\max} \in \{10, 20, 30\}$, column named sol contains the best objective function value obtained by the VNS in 20 runs, while column sol_{avg} represents the average objective function value in 20 runs. For each instance the best values of sol and sol_{avg} are bolded. The analysis of the obtained results shows that values of $k_{\max}$ and p influence the solution quality. For example, for instances 50_2 and 50_3, the best values of sol_{avg} are obtained for $k_{\max} = 10$ and $p = 0.1$, while for instance 300_3, the best sol_{avg} is for $k_{\max} = 20$ and $p = 0.5$. On the other hand, the best values of sol for instances 300_1, 400_2, 400_3 and 500_2 are obtained for combinations $k_{\max} = 30$ and $p = 0.5$, $k_{\max} = 30$ and $p = 0.1$, $k_{\max} = 20$ and $p = 0.5$, $k_{\max} = 30$ and $p = 0.5$, respectively. However, from Table 1 it follows that for combination $k_{\max} = 30$ and $p = 0$, the number of instances where the VNS achieved the best value for sol and sol_{avg} is 14 and 9, respectively. For all other combinations these numbers are smaller. This indicates that the best combination of parameters for this

Table 1 Experiments with parameters $k_{\max}$ and p

Inst.	$p = 0.0$						$p = 0.1$					
	$k_{\max} = 10$		$k_{\max} = 20$		$k_{\max} = 30$		$k_{\max} = 10$		$k_{\max} = 20$		$k_{\max} = 30$	
	sol	sol_{avg}	sol	sol_{avg}	sol	sol_{avg}	sol	sol_{avg}	sol	sol_{avg}	sol	sol_{avg}
50_1	**1204.41**	**1206.71**	**1204.41**	1207.33	**1204.41**	1219.81	**1204.41**	1218.19	**1204.41**	1223.23	**1204.41**	1219.64
50_2	**1340.44**	1348.09	**1340.44**	1350.82	**1340.44**	1357.93	**1340.44**	**1345.75**	**1340.44**	1355.41	**1340.44**	1357.81
50_3	**1316.39**	1321.84	**1316.39**	1339.81	**1316.39**	1340.48	**1316.39**	**1319.15**	**1316.39**	1323.96	**1316.39**	1338.00
100_1	**1217.47**	**1217.47**	**1217.47**	1217.54	**1217.47**	1217.51	**1217.47**	1336.25	**1217.47**	1493.98	**1217.47**	1217.51
100_2	**1128.40**	1128.50	**1128.40**	**1128.40**	**1128.40**	1128.46	**1128.40**	1135.47	**1128.40**	1143.38	**1128.40**	1128.46
100_3	**1252.99**	1254.20	**1252.99**	1253.34	**1252.99**	**1253.26**	**1252.99**	1264.40	**1252.99**	1258.58	**1252.99**	**1253.26**
200_1	**1206.79**	**1206.79**	**1206.79**	**1206.79**	**1206.79**	**1206.79**	**1206.79**	1212.00	**1206.79**	1220.63	**1206.79**	**1206.79**
200_2	**1213.24**	1217.24	**1213.24**	**1216.58**	**1213.24**	1217.16	**1213.24**	1226.40	1213.81	1236.44	**1213.24**	1217.08
200_3	**1247.25**	1248.63	**1247.25**	**1248.38**	**1247.25**	1248.43	**1247.25**	1260.49	**1247.25**	1259.86	**1247.25**	1249.34
300_1	1222.94	1233.39	1215.95	**1228.79**	1220.83	1231.32	1226.11	1237.60	1224.73	1235.12	1226.11	1236.79
300_2	**1170.85**	1179.66	**1170.85**	1178.15	**1170.85**	**1176.66**	**1170.85**	1181.58	**1170.85**	1189.09	**1170.85**	1180.57
300_3	**1247.51**	1268.08	**1247.51**	1266.20	**1247.51**	1263.81	**1247.51**	1273.22	1249.54	1275.90	1249.54	1274.36
400_1	1211.57	1231.81	**1211.33**	1226.30	**1211.33**	**1225.28**	1212.41	1243.11	1211.39	1235.84	1211.39	1235.08
400_2	1201.16	1216.62	1201.22	1213.55	1199.95	**1210.99**	1199.01	1221.22	1201.16	1220.81	**1197.66**	1218.52
400_3	1254.47	1272.04	1252.22	1267.76	1252.87	**1262.48**	1257.45	1277.71	1257.58	1270.81	1250.06	1271.23
500_1	1216.99	1246.66	1218.53	1239.50	**1205.82**	**1237.82**	1222.27	1253.36	1222.27	1253.46	1226.51	1243.23
500_2	1227.17	1253.74	1226.84	1253.39	1226.84	**1246.46**	1231.90	1261.28	1228.03	1258.74	1232.96	1255.33
500_3	1251.62	1268.91	1233.26	1264.26	**1231.81**	**1257.90**	1251.65	1272.49	1242.29	1269.20	1243.94	1261.94

Table 1 continued

Inst.	$p = 0.2$						$p = 0.5$					
	$k_{\max} = 10$		$k_{\max} = 20$		$k_{\max} = 30$		$k_{\max} = 10$		$k_{\max} = 20$		$k_{\max} = 30$	
	sol	sol_{avg}	sol	sol_{avg}	sol	sol_{avg}	sol	sol_{avg}	sol	sol_{avg}	sol	sol_{avg}
50_1	**1204.41**	**1206.71**	**1204.41**	1207.33	**1204.41**	1219.81	**1204.41**	1216.34	**1204.41**	1223.23	**1204.41**	1218.65
50_2	**1340.44**	1348.09	**1340.44**	1350.82	**1340.44**	1357.93	**1340.44**	1346.19	**1340.44**	1349.22	**1340.44**	1351.76
50_3	**1316.39**	1321.84	**1316.39**	1339.81	**1316.39**	1340.48	**1316.39**	1320.32	**1316.39**	1329.03	**1316.39**	1332.69
100_1	**1217.47**	**1217.47**	**1217.47**	1217.60	**1217.47**	1217.51	**1217.47**	1310.22	**1217.47**	1235.01	**1217.47**	1294.71
100_2	**1128.40**	1128.52	**1128.40**	1128.44	**1128.40**	1128.47	**1128.40**	1136.84	**1128.40**	1140.35	**1128.40**	1134.44
100_3	**1252.99**	1254.23	**1252.99**	1253.49	**1252.99**	1253.29	**1252.99**	1261.91	**1252.99**	1256.42	**1252.99**	1254.72
200_1	**1206.79**	**1206.79**	**1206.79**	**1206.79**	**1206.79**	**1206.79**	**1206.79**	1223.02	**1206.79**	1229.31	**1206.79**	1214.66
200_2	**1213.24**	1217.78	**1213.24**	1217.24	**1213.24**	1217.50	**1213.24**	1223.93	**1213.24**	1219.81	**1213.24**	1221.72
200_3	**1247.25**	1250.14	**1247.25**	1249.61	**1247.25**	1250.32	**1247.25**	1267.85	**1247.25**	1254.60	**1247.25**	1249.40
300_1	1226.11	1235.95	1225.64	1231.17	1226.11	1235.18	1222.52	1240.27	1215.51	1237.40	**1215.48**	1230.59
300_2	**1170.85**	1183.30	**1170.85**	1181.26	**1170.85**	1177.84	**1170.85**	1188.16	**1170.85**	1188.60	**1170.85**	1181.23
300_3	**1247.51**	1269.58	**1247.51**	1269.50	1253.27	1268.77	**1247.51**	1274.03	**1247.51**	**1263.63**	**1247.51**	1281.45
400_1	1219.37	1235.59	1218.20	1235.19	**1211.33**	1228.70	1216.03	1240.50	**1211.33**	1232.73	1212.27	1232.14
400_2	1202.59	1218.59	1201.22	1214.56	1202.59	1212.96	1201.36	1225.83	1209.44	1225.94	1202.59	1218.52
400_3	1254.47	1274.82	1256.17	1271.61	1252.87	1268.42	1255.77	1276.44	**1249.62**	1270.91	1255.77	1275.38
500_1	1216.99	1252.24	1226.33	1247.65	1215.78	1247.28	1223.38	1257.04	1216.39	1242.18	1218.07	1249.25
500_2	1227.17	1255.00	1227.71	1255.40	1226.84	1251.26	1229.45	1262.10	1243.46	1261.07	**1225.92**	1259.40
500_3	1251.62	1273.86	1250.03	1270.64	**1231.81**	1263.99	1250.97	1276.49	1231.87	1262.66	1244.09	1265.23

set of instances is $k_{\max} = 30$ and $p = 0$. Therefore, we used these values in all other experiments.

In order to examine the average behavior of the VNS with parameters $k_{\min} = 2$, $k_{\max} = 30$, $p = 0$, $iter_{\max} = 1000$, $t_{\max} = 600s$, we constructed a set of randomly generated instances, which include graphs with different number of vertices and edges. The number of vertices vary from 10 to 300 and the number of edges from 15 to 1000. In the generation process, for each edge a randomly generated real number is selected from interval [1, 10] representing its weight. The adjacency matrix of a graph is randomly generated avoiding self-loops and more than one edge connecting the same two vertices. If a graph created in this way is not connected, the instance is ignored and a new instance is constructed. In order to achieve the diversity, for each graph size, a set of three different instances was generated with a different random seed. The generated instances can be found on http://poincare.matf.bg.ac.rs/~zdrazic/dtp.

Table 2 contains the VNS results on small size randomly generated instances. In order to verify the results we applied CPLEX to the integer linear programming formulation of the DTP, introduced in [13]. In the first column, $Inst$, the instance name is given, containing the information about its dimensions. For example, instance $dtp\_10\_15\_2$ contains 10 vertices and 15 edges. The last number in the instance name represents the ordinal number of the instance of that size. The next two columns contain the information obtained with CPLEX: opt column contains the optimal objective function value and $time$ column represents the running time used by CPLEX to finish its work. If CPLEX could not provide the result, the symbol "–" is written. In the following six columns, the information about the proposed VNS algorithm is given. The column named sol contains the best objective function value found by the VNS in 20

Table 2 Experimental results on small size random instances

$Inst.$	CPLEX		VNS					
	opt	$Time$	sol	sol_{avg}	σ (%)	$ANDV$	t (s)	t_{tot} (s)
$dtp\_10\_15\_0$	5.89	0.06	opt	5.89	0.00	4.00	<0.01	0.04
$dtp\_10\_15\_1$	14.42	0.10	opt	14.42	0.00	5.00	<0.01	0.04
$dtp\_10\_15\_2$	14.35	0.14	opt	14.35	0.00	4.00	<0.01	0.04
$dtp\_15\_20\_0$	18.87	0.40	opt	18.87	0.00	6.00	<0.01	0.10
$dtp\_15\_20\_1$	23.03	0.44	opt	23.03	0.00	6.00	<0.01	0.09
$dtp\_15\_20\_2$	24.95	0.56	opt	24.95	0.00	6.00	<0.01	0.10
$dtp\_15\_30\_0$	18.20	4.10	opt	18.20	0.00	5.00	<0.01	0.11
$dtp\_15\_30\_1$	8.32	6.41	opt	8.32	0.00	4.00	<0.01	0.08
$dtp\_15\_30\_2$	18.07	12.62	opt	18.07	0.00	6.00	0.02	0.09
$dtp\_20\_30\_0$	33.81	342.31	opt	33.81	0.00	9.00	0.01	0.22
$dtp\_20\_30\_1$	36.03	281.74	opt	36.03	0.00	8.00	0.03	0.20
$dtp\_20\_30\_2$	43.50	268.80	opt	43.50	0.00	10.00	0.02	0.23
$dtp\_20\_50\_0$	9.81	169.58	opt	9.81	0.00	5.00	0.01	0.17
$dtp\_20\_50\_1$	–	–	12.19	12.19	0.00	6.00	0.01	0.16
$dtp\_20\_50\_2$	17.42	4179.54	opt	17.42	0.00	6.00	0.02	0.20

runs. If this value is equal to the optimal solution (from opt column), we mark it as opt. The next two columns sol_{avg} and σ, contain the information on the average solution quality. Value sol_{avg} represents the average objective function value in 20 independent runs, while σ^2 is the corresponding mean squared error, i.e. $\sigma^2 = \frac{1}{20}\sum_{i=1}^{20}(err_i - err)^2$, where $err = \frac{1}{20}\sum_{i=1}^{20} err_i$, $err_i = 100 \times \frac{VNS_i - sol}{sol}$, and VNS_i is the VNS solution obtained in i-th run. Column ANDV contains the average number of vertices in the VNS solution for 20 runs. The last two columns contain the average execution time t, used to reach the best VNS solution and the average total execution time t_{tot}.

As it can be seen in Table 2, the VNS quickly reaches optimal solutions obtained by CPLEX in all cases. In case of larger instances, dtp_20_30 and dtp_20_50, the VNS execution time values are from 1500 to 20,000 times smaller than CPLEX time values. Note that for instance dtp_20_50_1 CPLEX failed to find any solution with "Out of memory" error message.

Table 3 contains results of the proposed VNS for large size randomly generated instances, which could be used as a base for future comparisons with other metaheuristic approaches. It is organized in a similar way as Table 2. Here we do not have columns regarding CPLEX because it was not able to solve these instances. The column t_{tot} is also omitted because, in all cases, the algorithm stops when time $t_{\max} = 600s$ is exceeded.

Table 3 Experimental results on large size random instances

$Inst.$	VNS				
	sol	sol_{avg}	σ (%)	$ANDV$	$t(s)$
dtp_100_150_0	152.57	154.61	0.74	45.00	294.95
dtp_100_150_1	192.21	194.22	1.32	46.25	286.39
dtp_100_150_2	146.34	148.35	1.06	43.75	245.61
dtp_100_200_0	135.04	136.41	1.12	37.40	333.91
dtp_100_200_1	91.88	92.03	0.36	36.70	133.19
dtp_100_200_2	115.93	117.11	1.38	41.10	372.14
dtp_200_400_0	306.06	343.95	5.04	112.60	565.14
dtp_200_400_1	303.53	331.10	4.21	104.65	559.42
dtp_200_400_2	274.37	289.51	3.34	105.35	550.36
dtp_200_600_0	132.49	150.39	6.61	73.75	553.69
dtp_200_600_1	162.92	198.21	10.45	92.50	556.62
dtp_200_600_2	139.08	154.36	9.57	67.50	520.87
dtp_300_600_0	471.69	494.62	2.57	161.70	538.95
dtp_300_600_1	494.91	542.46	2.80	176.75	544.27
dtp_300_600_2	500.72	535.30	3.16	177.45	533.80
dtp_300_1000_0	257.72	264.33	1.01	134.40	575.10
dtp_300_1000_1	242.79	325.16	9.27	162.15	530.51
dtp_300_1000_2	223.18	251.41	7.92	109.35	482.59

Table 4 presents a comparison of the proposed VNS approach with the results from [14], obtained by the artificial bee colony algorithm (ABC_DT) and the ant colony optimization algorithm (ACO_DT) on the set of instances from Table 1.

In order to provide a fair comparison between these algorithms, we performed additional numerical experiments with the following stopping criterion: The algorithm stops when the maximum CPU time $t_{\max}$ is exceeded, where $t_{\max}$ is determined as follows. For each instance, $t_{\max}$ is equal to the average execution time t of either ABC_DT or ACO_DT. We chose average execution time t of the algorithm which has obtained smaller value of sol. If both algorithms had the same value of sol, we chose the smaller t. Note that with such stopping criterion $t_{\max} = t_{tot}$.

In Table 4, for each algorithm, the results are organized in the same way as in the previous tables. The additional last column, BSV, for each instance contains the overall best known objective function value arising from Tables 1 and 4.

From Table 4 it follows that the average objective function value sol_{avg}, obtained by the VNS, is better than values sol_{avg} for both ABC_DT and ACO_DT in 7 out of 18 instances, including 5 out of 6 largest instances with 400 and 500 vertices. The VNS produced strictly better values of the best objective function value sol in 8 out of 18 cases, compared to ABC_DT and ACO_DT. The ABC_DT was strictly better than both the ACO_DT and the VNS in one case, and ACO_DT was never better than both ABC_DT and the VNS. In 4 cases all three algorithms obtained the same values of sol.

In order to verify the significance of the obtained computational results, we provide the statistical analysis with the best objective function values sol of ABC_DT, ACO_DT and VNS. Demšar [2] showed that when comparing the classifiers over multiple data sets, the non-parametric Friedman test should be preferred over ANOVA. It is easy to see that this statement can be extended to general metaheuristic approaches and not only to classifiers. The Friedman test showed that there was a statistically significant difference in the obtained computational results among three considered algorithms: $p = 0.034$ (<0.05). Post hoc Wilcoxon signed-rank test with Bonferroni adjustment showed that there is no statistical difference between ACO_DT and ABC_DT ($p = 0.345$). Yet, there is a statistical difference between algorithm pairs: VNS and ABC_DT ($p = 0.016$), VNS and ACO_DT ($p = 0.026$). Based on this results, one can conclude that the VNS outperforms both compared algorithms, and that there is no significant difference between ABC_DT and ACO_DT.

Computational results in Tables 3 and 4 indicate that the VNS algorithm could be successfully applied to real-world large-scale networks. Namely, the average execution time t, used to reach high-quality VNS solutions does not increase rapidly with the size of the problem. For example, for instances 400_1 to 400_3 in Table 4, the average value of t is 397.89 s, while for instances 500_1 to 500_3 the average value of t is 556.24 s.

4 Conclusion

This paper is devoted to the recently introduced dominating tree problem. The problem is solved by the VNS algorithm that uses the set of vertex permutations in order to generate feasible solutions. The corresponding neighborhood structures allow an effective shaking procedure, which successfully diversifies the search process. For small dimen-

Table 4 Comparison with the existing methods

Inst.	ABC_DT					ACO_DT					VNS ($p = 0.0, k_{\max} = 30$)						*BSV*
	sol	sol_{avg}	σ (%)	*ANDV*	*t* (s)	*sol*	sol_{avg}	σ (%)	*ANDV*	*t* (s)	*sol*	sol_{avg}	σ (%)	*ANDV*	*t* (s)	t_{tot}	
50_1	1204.41	1204.41	0.00	19.00	25.57	1204.41	1204.41	0.00	19.00	2.41	1204.41	1247.10	2.57	20.85	1.93	2.41	1204.41
50_2	1340.44	1340.44	0.00	21.00	21.46	1340.44	1340.44	0.00	21.00	4.18	1340.44	1358.73	1.56	23.25	2.86	4.18	1340.44
50_3	1316.39	1316.39	0.00	19.00	22.99	1316.39	1316.39	0.00	19.00	2.50	1316.39	1334.81	1.25	21.00	2.01	2.50	1316.39
100_1	1217.47	1218.15	0.69	18.45	28.64	1217.47	1217.47	0.00	19.00	12.71	1222.65	1247.89	1.83	23.90	7.78	12.71	1217.47
100_2	1128.40	1128.42	0.09	17.90	27.58	1152.85	1152.85	0.00	17.00	10.86	1128.40	1137.19	0.98	16.05	17.21	27.58	1128.40
100_3	1252.99	1253.14	0.23	19.70	28.39	1253.49	1253.49	0.00	19.00	8.96	1252.99	1257.58	0.65	19.35	16.04	28.39	1252.99
200_1	1206.79	1209.52	2.69	18.25	84.10	1206.79	1207.61	3.58	18.05	81.13	1206.79	1209.24	0.45	18.30	34.54	81.13	1206.79
200_2	1216.41	1219.74	2.15	18.90	87.78	1216.23	1217.73	2.61	17.65	78.72	1216.23	1220.22	0.24	19.05	40.73	78.72	1213.24
200_3	1253.02	1258.06	3.42	22.15	90.44	1247.25	1248.94	2.99	20.90	97.93	1247.25	1259.63	0.95	21.50	74.23	97.93	1247.25
300_1	1229.97	1237.47	2.89	21.75	145.17	1228.24	1243.70	9.71	22.85	352.89	1226.11	1236.09	0.65	20.05	229.72	352.89	1215.48
300_2	1182.52	1200.79	7.82	19.60	162.59	1176.45	1193.95	10.51	21.10	260.30	1170.85	1181.43	0.95	18.70	177.83	260.30	1170.85
300_3	1257.21	1271.20	6.74	20.50	145.75	1261.18	1276.75	9.27	24.60	251.91	1262.57	1279.01	0.87	21.15	113.83	145.75	1247.51
400_1	1223.61	1241.75	7.88	21.90	263.13	1220.62	1237.45	9.50	26.05	600.74	1211.33	1225.21	0.56	20.45	432.83	600.74	1211.33
400_2	1220.54	1235.29	6.97	22.45	249.39	1209.69	1246.14	21.41	24.40	591.44	1202.59	1212.25	0.66	20.15	386.49	591.44	1197.66
400_3	1266.41	1276.80	4.59	22.30	216.95	1254.10	1270.34	9.42	25.85	530.58	1252.87	1267.72	0.68	21.30	374.35	530.58	1249.62
500_1	1233.14	1241.60	4.56	21.40	379.72	1219.66	1240.05	9.17	26.50	1163.20	1200.06	1224.82	1.12	20.30	834.19	1163.20	1200.06
500_2	1245.59	1258.33	5.40	22.35	364.04	1273.86	1295.51	13.39	28.65	1031.81	1227.18	1260.31	1.42	20.45	258.57	364.04	1225.92
500_3	1249.17	1278.67	11.96	21.60	338.25	1232.71	1259.08	20.03	24.35	917.73	1231.81	1252.86	1.00	21.10	575.95	917.73	1231.81

sions, the VNS reaches optimal values obtained by CPLEX in all cases, while for large instances, it gives better results than the existing ant colony and bee colony approaches.

One possible extension of this research can be directed toward modifying this approach in order to solve similar dominating problems on graphs. The second extension could be a parallelization of the presented approach and its testing on powerful multiprocessor computers.

Acknowledgements This research was partially supported by Serbian Ministry of Science under the Grants 174010 and 174033. The authors are grateful to the referees and the editor for the constructive comments and suggestions that improved the presentation of the paper.

References

1. Arkin, E.M., Halldorsson, M.M., Hassin, R.: Approximating the tree and tour covers of a graph. Inf. Process. Lett. **47**(6), 275–282 (1993)
2. Demšar, J.: Statistical comparisons of classifiers over multiple data sets. J. Mach. Learn. Res. **7**, 1–30 (2006)
3. Fujito, T.: On approximability of the independent/connected edge dominating set problems. Inf. Process. Lett. **79**(6), 261–266 (2001)
4. Fujito, T.: How to Trim an MST: A 2-Approximation Algorithm for Minimum Cost Tree Cover, Automata, Languages and Programming, pp. 431–442. Springer, Berlin, Heidelberg (2006)
5. Guha, S., Khuller, S.: Approximation algorithms for connected dominating sets. Algorithmica **20**(4), 374–387 (1998)
6. Hansen, P., Mladenović, N., Moreno-Perez, J.: Variable neighborhood search methods and applications, (invited survey). 4OR: Q. J. Oper. Res. **6**, 319–360 (2008)
7. Hansen, P., Mladenović, N., Moreno-Perez, J.: Variable neighborhood search algorithms and applications. Ann. Oper. Res. **175**, 367–407 (2010)
8. Mladenović, N., Hansen, P.: Variable neighbourhood search. Comput. Oper. Res. **24**, 1097–1100 (1997)
9. Mladenović, N., Kratica, J., Kovačević-Vujčić, V., Čangalović, M.: Variable neighborhood search for metric dimension and minimal doubly set problems. EJOR **220**, 328–337 (2012)
10. Mladenović, N., Kratica, J., Kovačević-Vujčić, V., Čangalović, M.: Variable neighborhood search for the strong metric dimension problem. Electron. Notes Discrete Math. **39**, 51–57 (2012)
11. Park M., Wang C., Willson J., Thai M.T., Wu W., Farago A.: A dominating and absorbent set in a wireless ad-hoc network with different transmission ranges. In: Proceedings of the 8th ACM International Symposium on Mobile Ad hoc Networking and Computing, pp. 22–31 (2007)
12. Prim, R.C.: Shortest connection networks and some generalizations. Bell Syst. Tech. J. **36**(6), 1389–1401 (1957)
13. Shin, I., Shen, V., Thai, M.: On approximation of dominating tree in wireless sensor networks. Optim. Lett. **4**, 393–403 (2010)
14. Sundar, S., Singh, A.: New heuristic approaches for the dominating tree problem. Appl. Soft Comput. **13**(12), 4695–4703 (2013)
15. Thai, M.T., Wang, F., Liu, D., Zhu, S., Du, D.Z.: Connected dominating sets in wireless networks with different transmission ranges. IEEE Trans. Mobile Comput. **6**(7), 721–730 (2007)
16. Wan, P.-J., Alzoubi, K.M., Frieder, O.: Distributed construction on connected dominating set in wireless ad hoc networks. In: Proceedings of the Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies, vol. 3, pp. 1597–1604 (2002)