Research Article GU J Sci 37(2): 636-652 (2024) DOI: 10.35378/gujs.1243008

Gazi University

Journal of Science

http://dergipark.gov.tr/gujs

Efficient Algorithm for Dominating Set in Graph Theory Based on Fundamental Cut-Set

Furkan OZTEMIZ*, Ali KARCI

İnonu University, Faculty of Engineering, Software Engineering Department, Malatya, Türkiye

Highlights

- Detection of dominant nodes in graphs
- Effect of fundamental cut-set on dominant nodes
- An efficient custom spanning tree - KMax Tree

Article Info

Received: 26 Jan 2023
Accepted: 03 July 2023

Keywords

Graph Theory
Dominating Set
Dominant Node
Karci Centrality
KMax Tree

Abstract

Determining the minimum dominating set in connected graphs is one of the most difficult problems defined as NP-hard. In this problem, it is aimed to determine the important nodes that can influence all nodes via the minimum number of nodes on the graph. In this study, an efficient near-optimal algorithm showing a deterministic approach has been developed different from the approximation algorithms mentioned in the literature for discovering dominating set. The algorithm has $O(n^3)$ time complexity in determining the Dominating Set (DS). At the same time, the algorithm is an original algorithm whose solution is not random by using a fundamental cut-set. The DS algorithm consists of 3 basic phases. In the first phase of the algorithm, the algorithm that constructs the special spanning tree (Karci Max tree) of the graph is developed. In the second phase, the algorithm that finds the fundamental cut sets using the Kmax spanning tree is developed. In the last phase, Karci centrality node values are calculated with fundamental cut set and by using these Karci centrality node values, an algorithm has been developed to identify DS nodes. As a result of these three phases, the dominance values of the nodes on the graph and the DS nodes are calculated. The detected Karci centrality node values give priority to the node selection for determining the DS. All phases of the developed DS and Efficient node algorithms were coded in R programming language and the results were examined by running on sample graphs.

1. INTRODUCTION

Graph theory is among the popular methods for solving many complex problems. Graph theory increases its development and usage area due to the easy modelling of daily problems and successful results of effective solution methods. In this study, an effective algorithm developed for detecting dominant nodes and determining the dominating set (DS) is mentioned. The developed algorithm is designed to work on unweighted and undirected graphs. The dominant nodes indicate the dominance of the people or objects modelled on the graph over each other. However, MDS aims to connect all nodes in the graph with the least number of nodes selected on the graph [1]. Detection of MDS on graph has been defined as an NP-Hard problem. Solution of NP-Hard problems is a type of problem whose solution takes time and whose time complexity increases exponentially [2]. The proposed algorithm is an efficient dominating set algorithm that gives the near-optimum solution. Many different algorithms are available in the literature to solve this problem. The MDS algorithms have been developed to provide solutions to many different real-world problems. An approximation algorithm has been developed to identify the efficiency of search routes of a connected wireless network. The main purpose is to provide maintenance and routing of some nodes that break down in the network [3]. In another study, MDS was determined with a performance value of $\ln\delta + 2$ with a greedy approximation algorithm. Symbol δ here indicates the maximum node degree of the input

*Corresponding author, e-mail: furkan.oztemiz@inonu.edu.tr graph [4]. By applying the MDS problem in Hopfield networks, successful results specific to artificial neural networks have been achieved. It is stated that the developed algorithm will also provide a solution to maximum clique, maximum independent set problems [5]. In addition to Greedy approached MDS algorithms, many methods designed with metaheuristic methods have been developed. An algorithm called tournament selection in the genetic algorithm (GA) structure is presented for MDS. The use of tournament selection was combined with the best pheromone update using the ant colony (ACO) algorithm. It aimed to explore the search area at a higher level. It has been stated that ACO algorithm gives more successful results than GA [6]. In another metaheuristic study, a solution for the MDS problem was presented by using an ant colony algorithm with two steps. With the help of the proposed new approach, pheromone correction strategy was added. Thus the problem of trapping the single-step ACO at the local optimum based on the greedy approach is eliminated [7]. Claimed to be more successful than classical greedy approximation algorithms and mixed heuristics-based algorithms, an order-based randomized local search (RLS) algorithm has been developed. This algorithm has yielded successful results for MDS detection in real-world problems including the Barabasi Albert Model unit disk graph, scale-free graph, and two social networks [8]. An approximation layer MDS algorithm with $O(n^2)$ time correlation has been developed. This algorithm was compared with the fastest algorithm available as of the publication date of the study [9]. An algorithm is presented that is faster than the fastest MDS algorithm with asymptotically $\Omega(2^n)$ time complexity. The time complexity of this new algorithm is $O(1,81^n)$. (n represents the number of nodes) [10]. A Linear algorithm is suggested for MDS by considering the minimum number of intersections in a T tree. In the study, to identify whether there are two discrete MDS for arbitrary bipartite graphs was detected as NP-Hard [11]. MDS algorithms in the literature consist of approximate solutions and greedy algorithms with high time complexity. In this study, a deterministic effective method is suggested. With the developed algorithm it is aimed to solve the MDS problem in near optimum. The time complexity of the algorithm that we developed is $O(n^3)$. This time complexity is a rather successful result for near optimum DS compared to MDS algorithms with $O(X^n)$ complexity [12]. In another study that produces a solution in polynomial time, it is aimed to determine approximately dominating set sets in connected graphs. In addition, approaches to solving a few graph problems such as travelling salesman of DS members identified in the study are included [13]. A study of directed graphs first shows how to incrementally calculate a minimum dominating set in ark additions. Then, in arc deletions, a minimally dominant cluster compute state is reduced to insertion. Finally, a new limit was placed on the size of the minimum dominant clusters based on these results [14]. In [15], A dominating set method developed with a greedy approach on a connected wireless network has been published. The comparative results of the developed method and the success of the method are given. In [16], Some domination results such as independent domination, total domination, connected domination, doubly connected domination, restrained domination, strong domination and weak domination have been tested on topological charts. In addition, the proofs of these results are explained in detail in the study. In another study, an algorithm is proposed to find the minimum connected dominant set (MCDS) for the unit disk graph based on the calculation of the convex body of the sensor nodes. The algorithm (n^2logn) developed specifically for unit disk graphs has time complexity [17]. Shortening of the solution time of the problem will provide it to be used in many real-time systems and problems. When the usage areas of DS in connected graphs are examined, it is seen to provide significant gains in many areas such as social networks, transportation systems, telecommunication, defence industry, health systems, etc. [18–21].

When the methods in the literature are examined, it is seen that there is no effective MDS algorithm for all graph types. Generally, there are metaheuristic algorithms that offer solutions close to the minimum dominating set and algorithms designed with the greedy approach. While most of these algorithms provide approximate solutions, some of them are algorithms that require much more time for execution. In addition, methods that give optimum results are developed specifically for certain graph types. The method we presented gives the optimum result in certain special graphs, but it gives near-optimum results in any graph. Another achievement of the presented algorithm is that it selects the members of the dominating set in polynomial time. In order to verify the success of the algorithm, analysis operations were performed on different graphs accepted in the literature and successful results were obtained. The proposed algorithm is coded in the R programming language. The igraph library is used for the creation of graph architecture and calculations.

Table 1. *Comparisons of algorithms for dominating set problem*

Approach	Robust	Deterministic	Time complexity	Space complexity	Optimum	Same Result for all executions
Statistical	No	No	N.A.	Polynomial / Exponential	Optimal, not stable	No
Heuristic	No	No	N.A.	Polynomial	Optimal, not stable	No
Greedy	Yes	Yes	Polynomial	Polynomial	Optimal	Yes
Exact	Yes	Yes	Exponential	Exponential in recursion case	Optimum	Yes
Proposed Method	Yes	Yes	Polynomial	Polynomial	Optimal	Yes

Comparative features of the algorithms developed for the dominating set problem are given in Table 1. The proposed method produces performance and near-optimal results with the greedy approach. The proposed method uses a special spanning tree(Kmax tree) and fundamental cut-sets when detecting dominating set members. In the literature, a solution to the DS problem has not been sought before by using fundamental cut-sets and Kmax tree. For this reason, the presented study has a unique structure in terms of DS solution approach.

2. MATERIAL METHOD

Graphs have 2 significant parameters as vertices and edges. Graphs are expressed as G = (V, E), where V is the set of nodes, and the edges combine nodes (vertices), and E is the set of edges connecting the nodes to one another [22]. Graphs are used easily in modelling many problems. Nodes are used to indicate the objects belonging to the problem, and edges are used to indicate the strength of the connection between objects. The notion of minimum dominant set that we emphasize in this study is a rather popular and compelling problem type in graph theory. G= (V, E) graph given as non-directional consists of V nodes and E edges. Each node in a dominating set is expressed as $D \subseteq V$. $v \in V$ is a member of D or a neighbour of a node belonging to D. If the dominant set is formed with the minimum possible number of nodes, that dominant set is called Minimum Dominating Set (MDS) [23]. The proposed method in Figure 1 is summarized in 5 stages.

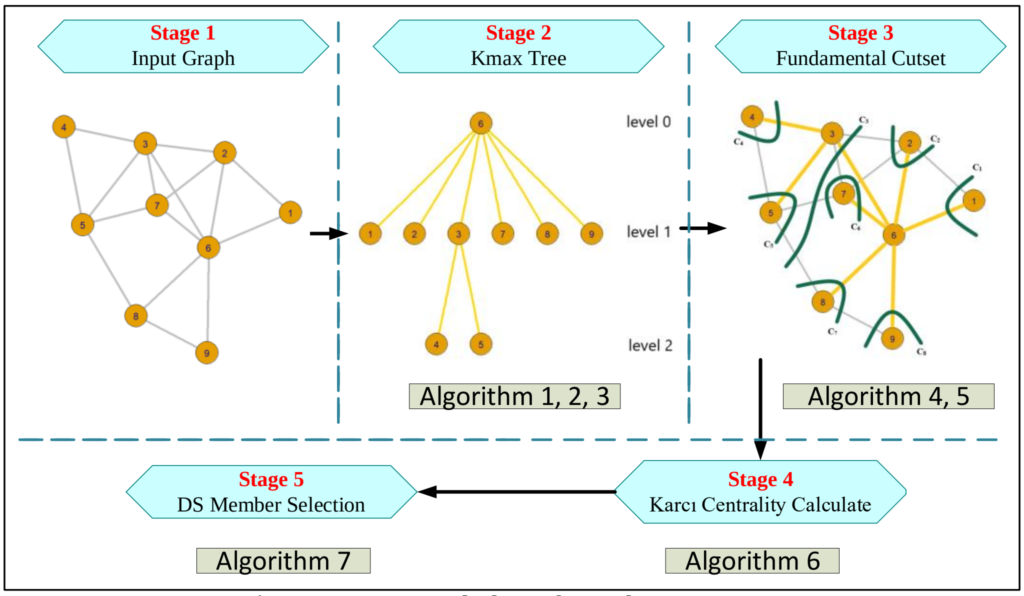


Figure 1. *Proposed algorithm solution process*

In Stage 1, the graph is brought into the appropriate form for analysis. In step 2, the Kmax tree of the graph is created. The edges of the Kmax tree represent the edges where the cuts will take place. In addition, Kmax degrees are also calculated at this stage. Algorithms 1,2, and 3 presented in the following sections are implemented at this stage. In step 3, fundamental cuts are made and fundamental cut-set degrees are calculated. Algorithms 4 and 5 are implemented at this stage. In Stage 4, the Karcı centrality value is calculated by applying Algorithm 6. In Stage 5, DS members are determined according to Karcı centrality values. Algorithm 7 is implemented at this stage.

2.1. Proposed Algorithms for Dominating Set

In order to determine the DS(Dominating Set) of the given graph, The Kmax tree must be constructed at first. The pseudo-code of the algorithm forming the Kmax tree is given in Algorithm 1.

Algorithm 1: Generating Kmax tree – O(n^3)

```
Kmax_Tree(A,AT,D)
1. Q←∅
2. r←max(D) // D is degree matrix
3. T=(V,E1) and E1←∅
4. i←1,…, |V|
5.     j←1,…,|V|
6.          AT(i,j)←0
7.  EnQueue(Q,r)
8.  Q2←∅
9. for i←1,…,|N(r)|
10.      EnQueue(Q2,$v_i$),    $v_i$∈N(r)
11.      if AT(r,s)=0 then
12.          AT(r,s)=1, AT(s,r)=1
13.          EnQueue(Q2,s)
14. while Q2≠∅
15.       v←DeQueue_Max(Q2,A,AT)
16.       EnQueue(Q,v)
17.      for i←1,…,|N(v)|
18.          if not($v_i$∈Q2) and $v_i$∈N(v)
19.              EnQueue(Q2,$v_i$)
20.              AT(v,$v_i$)=1, AT($v_i$,v)=1
21.      Remove_Zero(Q2,A,AT)
```

In Algorithm 1, one of the highest degree nodes in the graph is selected as the root node of the Kmax tree and its neighbour nodes are added to a queue, then the neighbour node degrees are reduced by one. Via DeQueue-Max(Algorithm 2), one of the nodes with the highest degree remaining degree from the nodes in the queue is selected as the next node to be expanded, and selected nodes are deleted directly from the queue. If there are more than one maximum nodes in the queue, the node selection is made according to queue order.

Algorithm 2: Selecting node whose remaining degree is maximum- O(n^2)

```
DeQueue_Max(Q,A,AT)
1. m←0
2. node←∅
3. i←1,…,|Q|
4.     v←DeQueue(Q)
5.     $t_1 \leftarrow \sum_{j=1}^{n} A(v,j)$, $t_2 \leftarrow \sum_{j=1}^{n} [A(v,j)\ and\ j \notin Q]$
6.     if m<|$t_1$-$t_2$| then
```

```
7.          m=t
8.          node=v
9.          EnQueue(Q,v)
10.     else EnQueue(Q,v)
11.     DeQueue(Q,node) // Removing “node” from Q.
12.   return node
```

Algorithm 2 chooses one of the nodes of which is in the queue and which has the maximum number of neighbours that is not at a high level. Kmax tree is constructed by using Algorithm 1 and Algorithm 2. In the 5th row of Algorithm 2, the function of [. . .] inside the total is a function of which result is 1 when the given condition is fulfilled and the result is 0 when it is not fulfilled. In order not to increase the number of elements in the queue unnecessarily, nodes of which the remaining degree is 0 are deleted from the queue. The algorithm that performs this process is given in Algorithm 3.

Algorithm 3: Remove nodes whose remaining degrees are zero – O(n^2)

```
RemoveZero(Q,A,AT)
1. i←1,…,|Q|
2.    v←DeQueue(Q)
3.    $t_1 \leftarrow \sum_{j=1}^{n} A(v,j),\ t_2 \leftarrow \sum_{j=1}^{n}[A(v,j)\ and\ j \notin Q]$

4.    if $t_1$-$t_2$≠0 then
5.        EnQueue(Q,v)
```

After the Kmax spanning tree is constructed, fundamental cut-sets must be obtained by using this spanning tree. Algorithm 4 is used to satisfy this aim. While fundamental cut-sets were being obtained, in the Kmax tree, edges that coincide with leaf nodes form the fundamental cut-set. The second fundamental cut-set type is the type of cut-set that divides the Kmax tree into two sub-trees. The first type of cut-set is obtained directly by Algorithm 4. The second type (internal cut-set) is obtained by Algorithm 5.

Algorithm 4: Generating fundamental cut-sets O(n^3).

```
FundamentalCutSets(AT,A,B,C)
1. B←IncidenceMatrix(A)        // size of A is nxn
2. for i←1,…, n
3.     for j←1,…,m             // size of B is nxm and number of edges is m
4.         C(I,j)=0
5. for i←1,…,m2
6.     EnQueue(E2,$e_i$)        // number of edges in T is m2 and T=(V,E2)
7. while E2≠∅
8.    e=DeQueue(Q2)
9.    if e=(u,v) and ($d_v$=1 or $d_u$=1)     //$d_u$ and $d_v$ illustrate degrees of u and v in T.
10.       if $d_u$=1 then
11.           i←1,…,m     // m is the number of edges in G=(V,E)
12.               C(u,i)=B(u,i)
13.    else GenerateInternalCut(AT,A,C,B,e)
```

Algorithm 5: Generating internal cut-sets – O(n^3)

```
GenerateInternalCut(AT,A,C,B,e)
1. Assume e=(u,v)
2. Q2←∅
3. EnQueue(Q2,u)
4. while Q2≠∅
5.     u←DeQueue(Q2)
6.     EnQueue(Q,u)
7.     for i←1,…,n                    //number of nodes in G=(V,E)
8.         if AT(u,i)=1 and i≠v
9.             EnQueue(Q2,i)
10. while Q≠∅
11.     u←DeQueue(Q)
12.     for i←1,…,n                    // n is the number of nodes in G=(V,E)
13.         if A(u,i)=1 and i∉Q
14.             C(u,k)=B(u,k)       // k illustrates the edge e=(u,i)
```

The aim of these algorithms (mentioned in previous sections) is to compute numeric value for each node. This value is called as Karci Centrality Value (KCV) in this study. The Karci centrality values of the nodes need to be calculated after the fundamental cut-sets are constituted. Algorithm 6 is recommended for this purpose. KCV refers to the dominance value of nodes. In the study, Karci centrality value was called as dominance value for each node. The process performed by this algorithm calculates the dominance values of all nodes. The node that the highest dominance value is the first element of DS. All the edges of the selected node in both the graph and the Kmax tree and all the edges of its neighbour nodes are deleted. The dominance value is recalculated according to the remaining Kmax in the next iteration. The node which has the highest dominance value is selected as the new DS member. While the DS members is selected, pendant node search in the graph is made in each iteration. If there is a pendant node in any iteration, the neighbour of the pendant node is selected as the DS node and then the same edge deletion process is applied. Unless there is a pendant node in the graph, the dominance values by using current Kmax and graph are calculated again. If there are nodes with the same dominance value, the node that has the lowest Kmax degree from among these nodes is selected as a DS member. The Algorithm 7 is used to detect pendant nodes.

Algorithm 6: Computing dominating number for each nodes- O(n^3)

```
1. Assume that G=(V,E) and T=Kmax_Tree(A,AT,D) where T=(V,$E_1$) is a Kmax tree of graph G.
2. B is the incidence matrix,of G and $C_{max}$ corresponds to Kmax.
3. $E_{max} = B * C_{max}^T$ where $E_{max}$ corresponds to Effectiveness and $C_{max}^T$ is the transpose of $C_{max}$.
4. i←1,…,n
5.     $\pi(v_i) = 0$
6.     j←1,…,m
7.         $\pi(v_i) = \pi(v_i) + E_{max}(i,j)$
8.         i←1,…,n
9.             $\pi(v_i) = \pi(v_i) + d_G(v_i) + d_T(v_i)$    where $d_G(v_i)$ is the node degree of $v_i$ in G, $d_T(v_i)$
   is the node degree in Kmax.
i.e. $\pi(v_i)$=Cut-Set Effectiveness+Graph Effectiveness+Spanning Effectiveness.
```

Algorithm 7: Pendant node- O(n^2)

```
Pendant(G,A,DS,D)
1. i←1,…,|V|
2.     if D($v_i$)=1 then
3.         DS=DS∪{$v_j$}  where N($v_i$)={$v_j$}
4.         k←1,…,|V|
5.             if A($v_i$,$v_k$)=1 then
6.                 A($v_i$,$v_k$)=0, A($v_k$,$v_i$)=0
```

The pseudo-codes of the stages of the proposed algorithm are given above. In Table 2, information about the time complexity of the algorithms is given.

Table 2. *Algorithms Time Complexities*

Algorithms	Time Complexities
Algorithm 1	O(n^3)
Algorithm 2	O(n^2)
Algorithm 3	O(n^2)
Algorithm 4	O(n^3)
Algorithm 5	O(n^3)
Algorithm 6	O(n^3)
Algorithm 7	O(n^2)
Algorithm General Notation	O(n^3)

3. THE RESEARCH FINDINGS AND DISCUSSION

Example 1: The proposed DS algorithm consists of a combination of many algorithms. The DS algorithm can be formed in two basic phases. In the first phase, whether there is a pendant node in the graph is checked. If there is a pendant node, its neighbour is chosen as the dominant node. If there are no pendant nodes in the second phase, the dominance value of the nodes is calculated. The node that the highest dominance value is chosen as the dominant node. The dominance values of the nodes on the graph consist of three important parameters. These parameters and the operating logic of the algorithm will be explained in detail through the example in Figure 2.

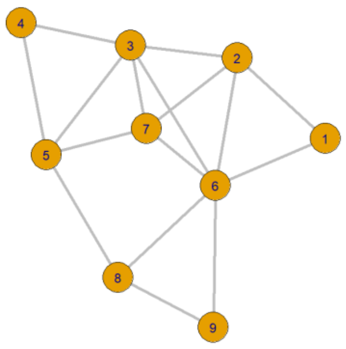


Figure 2. *Sample Graph*

If we explain node dominance value with a summary formula, it is calculated as follows; Node dominance values (Karci Centrality values) = Graf node degrees + Kmax tree node degrees + Fundamental cut-set degrees. First of all the node degrees of the example graph are calculated, and node degrees are expressed as the number of edges that the nodes have [24]. The node degrees of the example graph are shown in Table 3. In the second phase, the Kmax tree was set on the graph, and Kmax tree is a spanning tree that prioritizes nodes with high node degree [25]. While constructing the Kmax tree, neighbour nodes are reached by starting from the node whose degree is the highest degree on the graph. Then the node degrees of the reached neighbour nodes are decreased by one (The ancestor node and its edge is cut) and nodes that reached are put into the queue. The reason of neighbour nodes in queue is to set priority in selection for nodes whose node degrees are equal in subsequent iterations. According to the queue structure, first in - first out logic was applied. In the next iteration, while selecting the maximum node among neighbour nodes, if there is more than one node with maximum node degree, the first the level in the tree (the low level has priority) and then the order in the queue is considered. The selection priority is given to the node found first in the queue. If we explain the proposed method on the sample graph, when the node degrees of the sample graph is examined, it is seen that the node with the highest node degree is node 6. While Kmax tree is set, its neighbours are reached by starting from node 6 as shown in Figure 3. This situation constitutes the first phase of the Kmax tree. When the first phase is examined, the neighbours of the node 6 are (1, 2, 3, 7, 8 and 9) numbered nodes. When neighbour nodes are expanded in the Kmax tree, they are expanded according to index order.

Table 3. *Graph Node Degress*

Node Number	1	2	3	4	5	6	7	8	9
Node Degrees	2	4	5	2	4	6	4	3	2

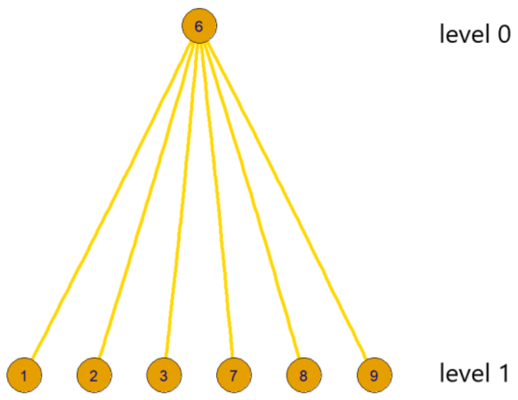


Figure 3. *Kmax tree 1st phase*

Table 4. *Current node degress of neighbour nodes*

Node Number	1	2	3	7	8	9
Node Degrees	0	0	2	1	1	0

The degrees of (1, 2, 3, 7, 8, and 9) numbered nodes are reduced by 1 before passing to the next iteration. (The edges of node 6 are cut). Also while the new degrees of nodes are being calculated, the edges of the nodes that connected to the previously visited nodes are deleted and the degrees of the nodes are recalculated. For example, the degree of node 7 is 4 according to the sample graph. In the Kmax tree, firstly its connection with the node 6 is deleted, and then the edges of these nodes with node 7 are deleted because of the node 2 and node 3 have reached before. Node 7 has only a connection with node 5. Therefore degree of node 7 is calculated as 1. This process is done at every iteration for all nodes that detected in Kmax. In this case, the updated node degrees are given in Table 4 before the 2nd iteration. According to Table 4, because the node 1, node 2 and node 9 are not neighbours to nodes that have not been reached before, their degrees are 0. There are two different nodes that node 3 can discover and one node that node 7 and node 8 discover. After the updated node degrees, new values are added to the queue as 1, 2, 3, 7, 8 and 9 numbered nodes, respectively (it is expressed in Table 5). Normally, node with zero degree is directly deleted from the queue, but in this study, to be understandable, it was not deleted in Table 5 and in Table 6.

Table 5. *1st iteration neighbour nodes queue structure*

Queue Rank	1	2	3	4	5	6
Node Number	1	2	3	7	8	9
Node Degrees	0	0	2	1	1	0

Table 6. *Queue structure of neighbouring nodes in 2nd iteration*

Queue Rank	1	2	3	4	5	6	7
Node Number	1	2	7	8	9	4	5
Node Degrees	0	0	0	0	0	0	0

1, 2, 3, 6, 7, 8, 9 numbered nodes have been reached in the graph. If all the nodes in the graph are reached, the algorithm will stop working. When the nodes in the neighbour nodes list are examined, the next maximum node is determined as the node 3. The node degree of the node 3 is 2, because there is no other node degree of 2 or more, in the neighbour nodes list, node 3 was selected. If there is another node whose degree is 2, firstly the node with a lower level in the Kmax tree and secondly node with a lower queue rank would be selected as the maximum node. The current neighbours of node 3 which the selected as the maximum node are node 4 and node 5. The connections of node 4 and node 5 are made with node 3 as shown in Figure 4. The degrees of node 4 and node 5 that newly reached are decreased by 1, and degrees of nodes in the queue are updated. The node 3 was selected as the maximum node, and it has been deleted from the queue list in Table 6. After the update process, degrees of all nodes in the queue given in Table 6 are 0. The process stops, since there are not new nodes to be reached and Kmax tree was formed. In this section, obtaining the fundamental cut-sets of graph by using Kmax tree will be given in detail. The node degrees of the Kmax tree, which is the second parameter of node dominance value algorithm, are given in Table 7. Kmax tree has another use. It is also used in calculating fundamental cut-set degrees, which is the 3rd parameter of the algorithm used in determining the dominant nodes. In Figure 5, Kmax tree is shown on the original graph.

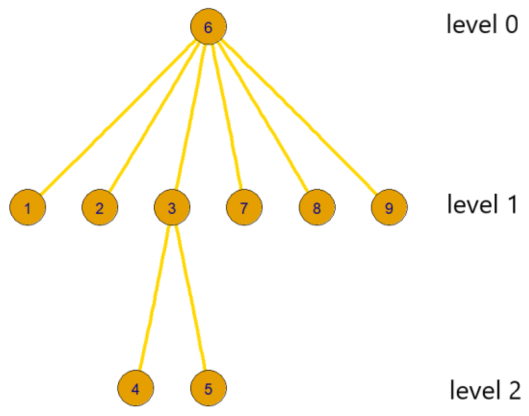


Figure 4. *Kmax tree 2nd phase*

Table 7. *Kmax tree node degree*

Node Number	1	2	3	4	5	6	7	8	9
Node Degrees	1	1	3	1	1	6	1	1	1

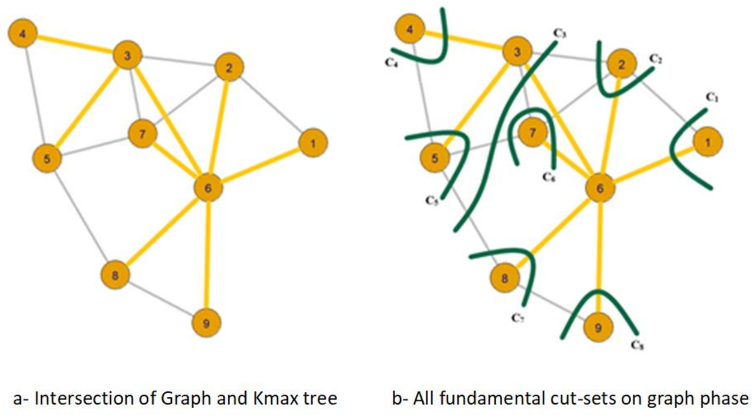

a- Intersection of Graph and Kmax tree
b- All fundamental cut-sets on graph phase

Figure 5. *Intersection of Graph and Kmax tree*

While calculating the fundamental cut-set degrees, all edges of the Kmax tree are deleted from the graph, respectively and the graph is divided into at least two subgraphs for every deletion process. By determining the edge relations among the nodes which belong to these two different clusters, these detected edges are cut and the affected nodes are determined. For instance, all fundamental cut-sets that were constructed for the graph structure shown in Figure 5a, are shown in Figure 5b. The fundamental cut-sets were constructed by cutting just one branch in Kmax tree and all the remaining cut edges are chords with respect to Kmax tree. Each fundamental cut-set divides Kmax tree into two sub-tree (divides graph into at least two subgraphs). When the 1st fundamental cut-set is examined, it separates the node 1 and node 6 in the Kmax tree. *n*th fundamental cut set was called as *Cn*. By this way, C1 (1st fundamental cut-set)={{1} , {2, 3, 4, 5, 6, 7, 8, 9}} numbered nodes construct two different groups. When the relations among these two groups are examined on the sample graph, the edge relations between 1-2 and 1-6 nodes are determined. The edge connection between node 1 and node 2 is stated with "1-2" expression. The fundamental cut-set of 1-2 and 1-6 connection gives 2 cut-set degree to node 1, 1 cut-set degree to node 2 and node 6. The cut-set degrees calculation from C1 to C8 cut-set will be explained in detail.

Two seperate node sets are constructed as C2={{2},{1,3,4,5,6,7,8,9}} numbered nodes, When the C2 given in Figure 5b is used. When the edge connections of these two sets with each other are examined, there are 1-2, 2-6, 2-7, 2-3 edges. In other words, if the C_2 occurs these edges will be broken off. After this cut-set operation, while 4 cut-set degrees are added for node 2, 1 cut-set degree is added to node 1, node 3, node 7, and node 6. The C_3 divides the graph into two sets as C_3 = {{3, 4, 5}, {1, 2, 6, 7, 8, 9}} numbered nodes. When Figure 5 is examined, this cut-set breaks off 3-2, 3-6, 3-7, 5- 7, 5-8 numbered connections. After this process, node 2 gained 1 cut-set degree while node 3 gained 3, node 5 gained 2, node 6 gained 1, node 7 gained 2 and node 8 gained 1. The C4 divides the graph into two sets as C4 = {{4}, {1, 2, 3, 5, 6, 7, 8, 9}} numbered nodes. With the occurrence of C_4, node 4 gained 2 cut-set degree while node 3 and node 5 gained 1. The C5 divides the graph into two sets as C_5 = {{5}, {1, 2, 3, 4, 6, 7, 8, 9}} numbered nodes. With the occurrence of C_5, node 5 gained 4 cut-set degree whereas node 3, node 4, and node 8 gained 1 cut-set degree. The C_6 divides the graph into two sets as C_6 = {{7}, {1, 2, 3, 4, 5, 6, 8, 9}} numbered nodes. With the occurrence of C_6, node 7 gained 4 cut-set degree, while node 2, node 3, and node 6 gained 1. The C_7 divides the graph into two sets as C_7 = {{8}, {1, 2, 3, 4, 5, 6, 7, 9}} numbered nodes. With the occurrence of C7, node 8 gained 3 cut-set degree, while node 5, node 6, and node 9 gained 1 cut-set degree. Finally, the C_8 divides the graph into two sets as C_8 = {{9}, {1, 2, 3, 4, 5, 6, 7, 8}} numbered nodes. With the occurrence of C_8, node 9 gained 2 cut-set degree whereas node 6 and node 8 gained 1 cut-set degree. Table 8 shows the all cut-set degrees until from C_1 to C_8. With the completion of all cut-set processes, the total node cut-set degrees were determined in Table 8. After determining the last parameter, the node dominance values were determined with the following formula (1). The node dominance values (Γ) of the sample graph are given in Table 9.

$$(\Gamma) = \text{Graph degrees} + \text{Kmax degrees} + \text{Cut} - \text{set degrees} \tag{1}$$

Table 8. *All fundamental cut-sets node degrees*

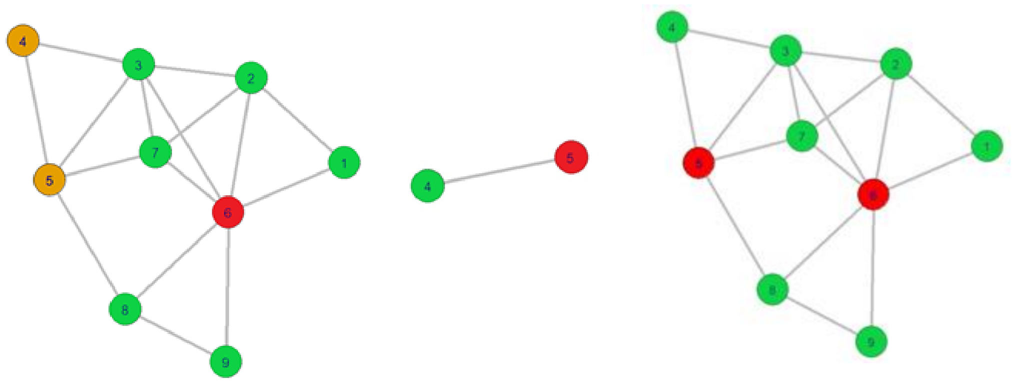
Nodes / Cut-sets	1	2	3	4	5	6	7	8	9
C_1	2	1	-	-	-	1	-	-	-
C_2	1	4	1	-	-	1	1	-	-
C_3	-	1	3	-	2	1	2	1	-
C_4	-	-	1	2	1	-	-	-	-
C_5	-	-	1	1	4	-	1	1	-
C_6	-	1	1	-	1	1	4	-	-
C_7	-	-	-	-	1	1	-	3	1
C_8	-	-	-	-	-	1	-	1	2
C_1- C_8	3	7	7	3	9	6	8	6	3

Table 9. *Node Dominance Values (KCV)*

Node Number	Γ (KCV)	Graph Degrees	Kmax Degrees	Cut-set Degrees
1	6	2	1	2
2	12	4	1	7
3	15	5	3	7

4	6	2	1	3
5	14	4	1	9
6	18	6	6	6
7	13	4	1	8
8	10	3	1	6
9	6	2	1	3

While determining the dominating set, it is checked whether there is a pendant node on the graph at the first phase. Nodes whose node degree is 1 are called Pendant nodes [26]. If there are pendant node or nodes in the graph, the ancestor of these nodes, in other words, its neighbour is determined as the dominant node. After the dominant node has been determined all edges of the dominant node, neighbours of the dominant node and edges of neighbours of the dominant node are deleted from both the graph and the Kmax tree. The graph and Kmax tree are updated. The dominant node is added as a DS member. If there is no pendant node, the node dominance value of the updated graph is calculated. The node with the highest dominance value is chosen as the dominant node. If there is more than one node that has the highest dominance degree, the node with lower Kmax tree degree is selected first. If there is still equality, any node can be chosen. The dominant node, the neighbours of the dominant node, and all the edges of the neighbours of the dominant node are deleted from the graph and the Kmax tree, then the graph and Kmax are updated. The detected dominant node is added as a DS member to DS list. The pendant node control is performed at the beginning of each iteration. If there is a pendant node in any iteration on the graph, the 1*st* phase is performed. In all iterations where there is not pendant node, the 2*nd* phase is performed to determine the DS members. The process continues until all the nodes in the graph are finished. If we explain from the example graph, there is not pendant node in the graph, so the 2*nd* phase is applied. The node with the highest dominance value in the graph is node 6. Therefore, its neighbours were detected by determining the node 6 as the dominant node. Figure 6a shows the 1st iteration of the DS algorithm visually. As it can be understood from the figure, with the selection of the node 6, the access is provided to 1,2,3,7,8,9 numbered nodes. The reached nodes must be removed from the graph with their edge connections. Likewise, The Kmax tree is updated by deleting the necessary connections. If we examine Figure 6a, we can see that red nodes represent dominant nodes, green nodes represent neighbour nodes of dominant nodes, orange nodes represent nodes that haven't been reached yet.

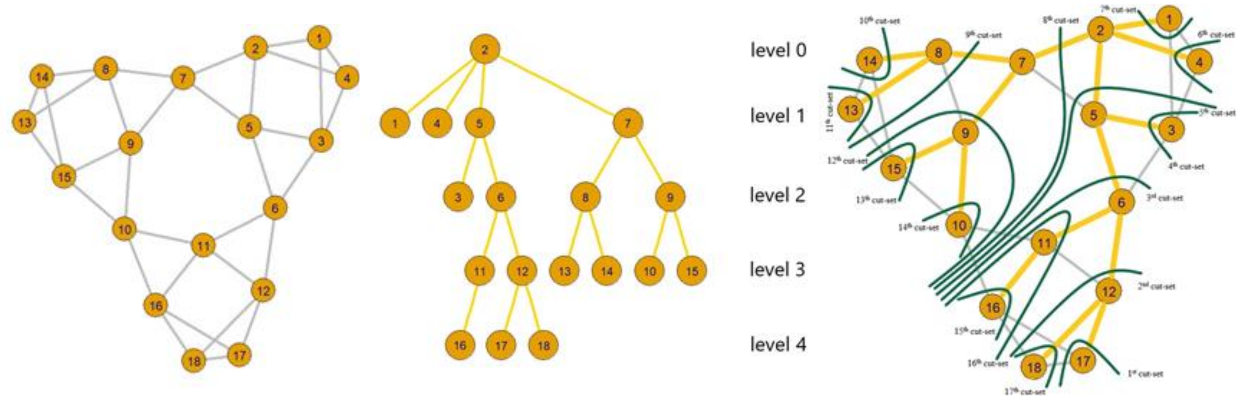


a- DS algorithm 1st iteration b- DS algorithm 2nd iteration c- Dominating Set

Figure 6. *Dominatig Set members*

The node 6 and its neighbour nodes (1, 2, 3, 7, 8, 9 numbered nodes) and the edge connection of these nodes have been deleted from the graph. After the deletion, node 4 and node 5 which were not reached before remained in the graph. A special case has been encountered here. As shown in Figure 6b, the pendant node feature cannot be used due to the fact that node degree of both nodes is 1. By applying second phase for the solution, node dominance values are calculated for Figure 6b. The dominance values of node 4 and node 5 were determined as 1. This value results from the degree of node, because there is not an edge connecting two nodes in Kmax tree. Therefore, the cut-set degree does not occur. Normally, in case of equality in node dominance value, the Kmax tree degrees of these nodes are checked. For Kmax tree degrees cannot be calculated, either of the two can be chosen as the dominant node. Considering that node 5 is chosen as the 2nd dominant node, for node 5, and node 6 and their neighbours form the whole graph, this set with 2 elements is called Dominating Set. The node 5 and node 6 are the dominating set elements since all nodes in the graph cannot be reached with less than 2 nodes. In Figure 6c, the red coloured nodes show the DS nodes and the green coloured nodes show the neighbour nodes that are reached through the DS nodes.

Example 2: In the previous sections, the node dominance values and the phases of the DS algorithms were explained in detail. In this section, DS has been calculated on a claw-free graph and a randomly generated sample graph by explaining in brief. A claw-free graph consisting of 18 nodes and 33 edges will be called as graph A. Graph A is shown in Figure 7a.



a- Graph A b- Graph A Kmax tree c- All cut-sets on graph A

Figure 7. *Graph A Proposed algorithm process*

There is no pendant node in graph A. Therefore, the node dominance value of the graph A must be calculated to determine the first DS member. All fundamental cut-sets and Kmax tree of the graph A are given in Figure 7b and Figure 7c. Also the node degrees, the Kmax tree node degrees, the fundamental cut-set degrees and the node dominance values of graph A are given in Table 10. After calculating the node dominance values of the graph A, DS calculation was started, because there is not a pendant node of graph A in the 1st iteration, the node dominance values are examined.

The node 16 and node 10, whose node dominance value is 21, have the highest node dominance values. One of these two nodes can be selected. In this study, the first member of DS was chosen as the node 16. In Figure 8a. first dominant node (node 16) was highlighted in red. Its neighbours were highlighted as green. The structure was updated by deleting the red and green nodes and all edge connections of these nodes from both graph A and Kmax trees. When the pendant node control was performed in the 2*nd* iteration, it was determined that the node 12 given in Figure 8b was a pendant. According to the algorithm, the node 6 which is the ancestor of the node 12 have been selected as the dominant node and added to the DS. With red and green nodes (node 6 and neighbours of node 6), and all edge connections of these nodes are deleted from graph A and Kmax tree. The current graph in the 3*rd* iteration has been given in Figure 8c. There is not pendant node on the graph. For this case, the node dominance values are recalculated for Figure 8c with the updated Kmax tree.

Table 10. *The node dominance values (Karci centrality values) of Graph A*

Node Number	Γ (KCV)	Graph Degrees	Kmax Degrees	Cut-set Degrees
1	10	3	1	6
2	12	4	4	4
3	14	4	1	9

4	10	3	1	6
5	12	4	3	5
6	12	4	3	5
7	12	4	3	5
8	12	4	3	5
9	12	4	3	5
10	21	4	1	16
11	16	4	2	10
12	12	4	3	5
13	11	3	1	7
14	11	3	1	7
15	16	4	1	11
16	21	4	1	16
17	11	3	1	7
18	11	3	1	7

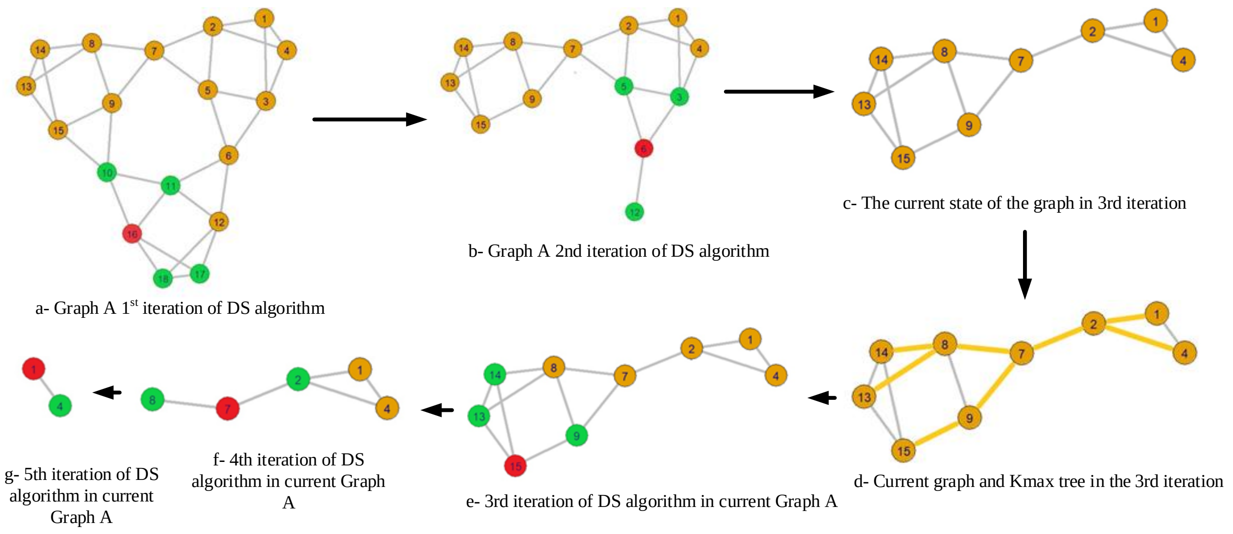

a- Graph A 1st iteration of DS algorithm
b- Graph A 2nd iteration of DS algorithm
c- The current state of the graph in 3rd iteration
d- Current graph and Kmax tree in the 3rd iteration
e- 3rd iteration of DS algorithm in current Graph A
f- 4th iteration of DS algorithm in current Graph A
g- 5th iteration of DS algorithm in current Graph A

Figure 8. *Graph A dominating set members selection process*

Figure 8d shows that current graph A and current Kmax tree in the 3rd iteration. As a result of the calculations, the node dominance values of graph in Figure 8c are given in Table 11.

Table 11. *The node dominance values of Figure 8c*

Node Number	Γ (KCV)	Graph Degrees	Kmax Degrees	Cut-set Degrees
1	6	2	1	3
2	9	3	3	3
4	6	2	1	3
7	9	3	3	3
8	12	4	3	5
9	9	3	2	4
13	11	3	1	7
14	11	3	1	7
15	13	3	1	9

When Table 11 is examined, the highest node dominance value belongs to node 15. As shown in Figure 8e, node 15 is selected as the third member of the DS. The node 15 and all the edges of the neighbours of node 15 are deleted on the graph A and Kmax tree. Graph and Kmax tree are updated.

In the 4*th* iteration, pendant node control is made. As shown in Figure 8f, node 8 is a pendant node, so its ancestor (its neighbour) is chosen as the dominant node. After the node 7 is added as a DS member, the required deletion is done on the graph and the graph is updated.

As shown in Figure 8g, we cannot apply the pendant node rule in the 5*th* iteration because the node degree of the remaining two nodes is 1. The node dominance values of these nodes are also equal. For DS selection, selection of any of the two nodes is sufficient. Finally, node 1 was selected for DS. The dominating set nodes were determined as 16, 6, 15, 7, and 1 numbered nodes, respectively. If node 10 was chosen instead of node 16 in the 1st iteration, DS nodes would be 10, 3, 7, 13, and 17 numbered nodes. In both options, number of the DS is 5 nodes.

The algorithm was run on complex network given in Figure 9 and successful results were obtained. While the red nodes given in Figure 9 are DS nodes, the green nodes represent the neighbours of the DS nodes. By applying the algorithm to the complex network, 17 dominant nodes were determined for DS. The way how these 17 nodes were detected is shown in Table 12.

Table 12. *Complex Network DS nodes detection methods*

Iteration	DS Node Number	Detect Methods
1	60	Dominance Value
2	39	Pendant ancestor
3	37	Pendant ancestor
4	7	Pendant ancestor
5	63	Dominance Value
6	36	Lonely Node
7	10	Pendant ancestor
8	56	Dominance Value
9	52	Pendant ancestor
10	67	Pendant ancestor
11	45	Dominance Value
12	12	Pendant ancestor
13	24	Pendant ancestor
14	4	Dominance Value
15	16	Pendant ancestor
16	26	Pendant ancestor
17	34	Dominance Value

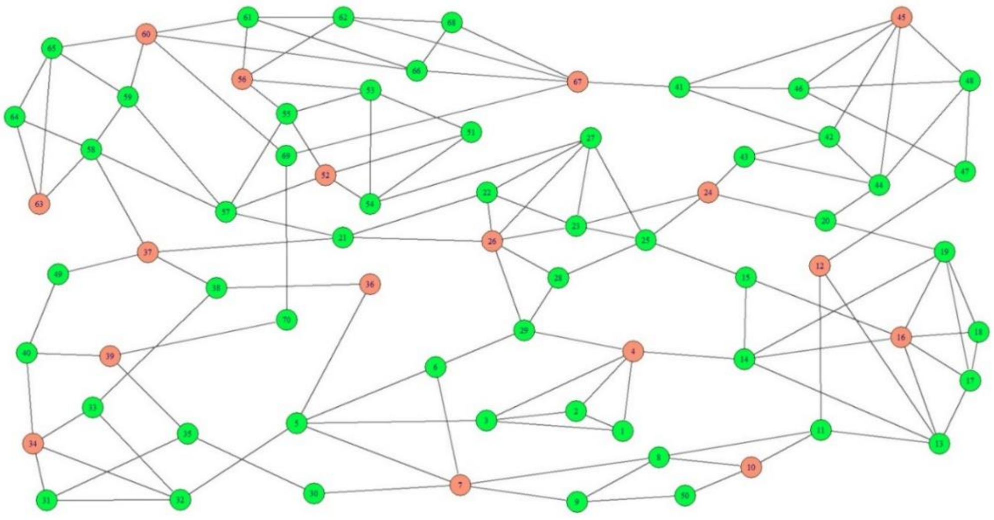


Figure 9. *All dominating and neighbour nodes belonging to graph B*

The proposed algorithm has also been tested in the well-known social network[27] datasets in the literature. In Figure 10, analysis results of zachary karate club, zebra and dolphin networks are given.

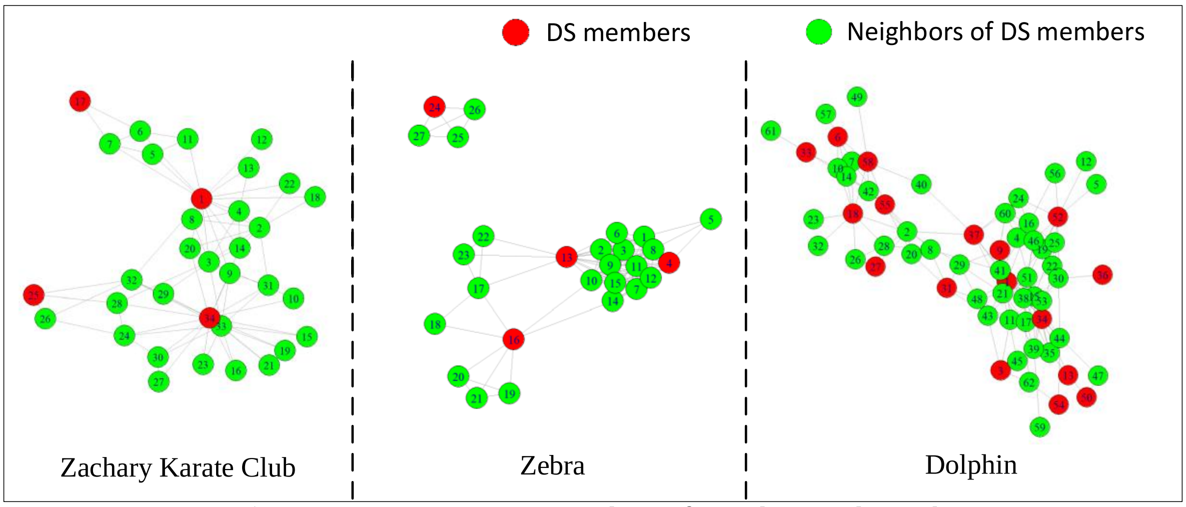


Figure 10. *Dominating set members of social network graphs*

Zachary karate network has 34 nodes and 78 edge connections. As a result of the application of the proposed method, the number of dominating set members was determined as 4. In Figure 10, DS members are highlighted in red and their neighbours are highlighted in green. For Zebra network with 27 nodes and 111 edge connections, the dominating set value was determined as 4. For the dolphin network with 62 nodes and 159 edges, the number of dominating set members is set to 17. When the results are examined, the DS values determined are generally optimum, but with the worst probability, they are optimal.

4. RESULTS

In this study, an effective optimal DS algorithm was developed to approximate solution the MDS problem known as NP-Hard in graph theory. The developed algorithm is deterministic and it has high performance. The algorithm was applied step by step in different graph types and successful results were obtained. To demonstrate the effectiveness of the algorithm, it has been tested on the well-known social network datasets zachary karate club, zebra and dolphin networks. When the test results of the proposed method are examined, it is determined that the optimum solution is generally reached, but in some graphs, near-optimal solution is determined. Therefore, the proposed algorithm guarantees the optimal dominating set for any graph type. The proposed algorithm has a unique structure that does not use randomness and where fundamental cut sets are used effectively in the DS problem. All phases of the developed dominating set algorithm are explained in detail in this study. The process of constructing the Kmax tree and fundamental cut-sets, which form the most unique and complex parts of the algorithm is explained step by step. In addition, a Pseudo code for the algorithm is given in this study. As stated in the pseudo-code, the DS algorithm has O(n^3) notation which is highest time complexity value. In addition, the time complexity values of the important stages of the proposed method are included in the study. It is anticipated that the proposed algorithm can be used in the solution of independent set and clique problems with its developable structure.

The proposed algorithms are subject to revision in case of efficiency and validity. We will take in consideration the efficiency and analytic proofs of these algorithms in future research.

CONFLICTS OF INTEREST

No conflict of interest was declared by the authors.

REFERENCES

[1] Shen, C., and Li, T., “Multi-document summarization via the minimum dominating set”, In Proceedings of the 23rd International Conference on Computational Linguistics (COLING ’10), 984-992, (2010).

[2] Fomin, F., Kratsch, V. D., Woeginger, G.J., “Exact (Exponential) Algorithms for the Dominating Set Problem”, Graph-Theoretic Concepts in Computer Science, 3353: 245–256, (2004).

[3] Wang, N., Dai, J., Li D., and Li, M., “An approximation algorithm for connected dominating set in wireless ad hoc network”, (ICISCE 2012). 1–5, (2012).

[4] Ruan, L., Du, H., Jia, X., Wu, W., Li, Y., and Ko, K., “A greedy approximation for minimum connected dominating sets”, Theoretical Computer Science, 329(1-3): 325–330, (2004).

[5] Xu, X., Tang, Z., Sun, W., Chen, X., Li, Y., Xia, G., Bi, X., and Zong, Z., “An algorithm for the minimum dominating set problem based on a new energy function”, SICE 2004 Annual Conference, 924– 926, (2004).

[6] Ho, K.C., Singh, Y. P., and Ewe, H.T., “An Enhanced Ant Colony Optimization Metaheuristic for The Minimum Dominating Set Problem”, Applied Artificial Intelligence, 881–903, (2006).

[7] Jovanovic, R., and Tuba, M., “Ant Colony Optimization Algorithm with Pheromone Correction Strategy for the Minimum Connected Dominating Set Problem”, Computer Science and Information Systems, 10(1): 133–149, (2013).

[8] Chalupa, D., “An order-based algorithm for minimum dominating set with application in graph mining”, Information Sciences, 426: 101–116, (2018).

[9] Zhou, Y., Lv, G., Xiu, B., Zhang, W., and Cheng, Q., “A faster approximate method to identify minimum dominating set”, IEEE 5th International Conference on Software Engineering and Service Science, 443–448, (2014).

[10] Grandoni, F., “A note on the complexity of minimum dominating set”, Journal of Discrete Algorithms, 4(2): 209–214, (2006).

[11] Grinstead, D. L., and Slater, P. J., “On minimum dominating sets with minimum intersection”, Discrete Mathematics, 86(1-3): 239–254, (1990).

[12] Campan, A., Truta, T.M., and Beckerich, M., “Fast dominating set algorithms for social networks”, CEUR Workshop Proceedings, 1353, 55–62, (2015).

[13] Guha, S., Khuller, S., “Approximation Algorithms for Connected Dominating Sets”, Algorithmica, 20: 374–387, (1998).

[14] Chaoyi, P., Rui Z., Qing, Z., Junhu, W., “Dominating sets in directed graphs”, Information Sciences, 180(19): 3647-3652, (2010).

[15] Yingshu, L., My, T., Thai, Feng, W., Chih-Wei, Y., Peng-Jun. W., Ding-Zhu, Du., “On greedy construction of connected dominating sets in wireless networks”, Wireless Communications and Mobile Computing, 5: 927–932, (2005).

[16] Jwair, Z., Abdlhusein, M., “Some dominating results of the topological graph”, International Journal of Nonlinear Analysis and Applications, 1-9, (2022).

[17] Purohit, G., N., Sharma, U., “Constructing Minimum Connected Dominating Set: Algorithmic approach”, International journal on applications of graph theory in wireless ad hoc networks and sensor networks (GRAPH-HOC), 2(3): 59-66, (2010).

[18] Truta, T. M., Campan, A., Beckerich, M., “Efficient Approximation Algorithms for Minimum Dominating Sets in Social Networks”, International Journal of Service Science, Management, Engineering and Technology (IJSSMET), 9(2): 1–32, (2018).

[19] Cao, H., Wu, W., Chen, Y., “A navigation route based minimum dominating set algorithm in VANETs”, International Conference on Smart Computing Workshops, 71–76, (2014).

[20] Balasundaram, B., and Butenko, S., “Graph Domination, Coloring and Cliques in Telecommunications”, Handbook of Optimization in Telecommunications, 865–890, (2006).

[21] Shen, C., and Li, T., “Multi-document summarization via the minimum dominating set”, In Proceedings of the 23rd International Conference on Computational Linguistics, (COLING ’10), Association for Computational Linguistics USA, 984–992, (2010).

[22] Öztemiz, F., Karcı, A., “Akademik Yazarların Yayınları Arasındaki İlişkinin Sosyal Ag Benzerlik Yöntemleri İle Tespit Edilmesi”, Uludag University Journal of The Faculty of Engineering, 25(1): 591–608, (2020).

[23] Potluri, A., Singh, A., “Hybrid metaheuristic algorithms for minimum weight dominating set”, Applied Soft Computing, 13(1): 76–88, (2013).

[24] Kapoor, K., Sharma, D., Srivastava, J., “Weighted node degree centrality for hypergraphs”, IEEE 2nd Network Science Workshop, 152–155, (2013).

[25] Karci, A., “New Algorithms for Minimum Dominating Set in Any Graphs”, Anatolian Science, 5(2): 62–70, (2020).

[26] Uehara, R., Toda, S., Nagoya, T., “Graph isomorphism completeness for chordal bipartite graphs strongly chordal graphs”, Discrete Applied Mathematics, 145(3): 479–482, (2005).

[27] http://konect.cc/. Access date: 24.05.2023