ALGORITHMIC APPROACH FOR DOMINATION NUMBER OF UNICYCLIC GRAPHS

Dr. S. V. Shindhe

Assistant Professor, Department of Computer Science, Karnatak University, Dharwad.

ABSTRACT

Let $G(V, E)$ be a unicyclic graph. A unicyclic graph is a connected graph that contains exactly one cycle. A dominating set of a graph $G = (V, E)$ is a subset D of V , such that every vertex which is not in D is adjacent to at least one member of D . The domination number is the number of vertices in a smallest dominating set for G . In this paper I have presented an algorithmic approach to compute the domination number and the minimum domination set for the unicyclic graph. The algorithm has polynomial time complexity of $O(n)$.

Key words: Graph, domination, minimum domination, unicyclic graph, algorithm, polynomial.

Cite this Article: Dr. S. V. Shindhe, Algorithmic Approach for Domination Number of Unicyclic Graphs, *International Journal of Computer Engineering and Technology*, 10(2), 2019, pp. 49-59.

<http://iaeme.com/Home/issue/IJCET?Volume=10&Issue=2>

1. INTRODUCTION

The concept of dominations is used naturally in many situations like war planning, some chess board games like n-queens and in some rural games of India. Finding the minimum domination number or minimum domination set for an arbitrary graph is NP-Complete. An algorithm was presented in [3] by E. Cockayne, S. Goodman and S. Hedetniemi for finding the minimum dominating number for trees, in polynomial time. The same algorithm is modified in this paper and presented a polynomial time algorithm for determining the minimum domination number for unicyclic graph.

2. BASIC TERMINOLOGY

Some of the basic graph terminologies used in this paper are as below.

- Graph: A graph $G = (V, E)$ is an ordered pair of sets. Where the elements of V are called vertices or nodes, and elements of $E \subseteq V \times V$ are called edges or lines. We refer to V as the vertex set of G , with E being the edge set.

- Tree and Unicyclic Graph: Tree is a graph without any cycles in it. It is also called acyclic tree. A graph which has only one cycle in it is called Unicyclic Graph.
- Vertex Domination: The subset of S of V is a dominating set of G if every vertex outside S has a neighbour in S , i.e., $N[S] = V$. The domination number is the smallest size of a dominating set of G . The domination number of a graph G is also said to be the minimum cardinality of a dominating set of G .

3. MINIMUM DOMINATION NUMBER OF UNICYCLIC GRAPH:

Let the vertices of unicyclic graph be partitioned into three types as: free vertices, bound vertices and required vertices. The free vertices need not be dominated but however may be included in dominating set to dominate the bound vertices. The required vertices constitute the minimum dominating set. The bound vertices are dominated by required vertices or we can say that these are the vertices to be dominated. Initially all the vertices of unicyclic graph are labelled as bound vertices.

The construction of minimum dominating set for of unicyclic graph can be done at two stages. In the first stage we consider only the tree part of the graph and find the minimum dominating set for it. However some of the vertices of the cycle may be visited. In the second stage we consider the cycle part of the graph and find the minimum dominating set for it. Finally the dominating set of unicyclic graph is given by,

$$\text{mindomset}(UCG_n) = \text{mindomset}(T_m) \cup \text{mindomset}(C_k)$$

Where m is the number of vertices of tree part T_m and k is the number of vertices in the cycle part C_k such that $n = m + k$.

3.1. Method for tree part:

The algorithm starts with finding all the end vertices of the unicyclic graph. In this, it has only the vertices of tree with degree one. Each end vertex is added to a queue (or an array). Each of the end vertex v is taken (deleted) from queue and its adjacent vertex u is obtained. The vertex u can be marked as free, bound or required based on the following conditions.

- If v is bound vertex then u is marked as required, so that v is dominated.
- If v is free vertex then u is marked as bound vertex, meaning that u must be dominated later. We can mark the last free vertex as required only if it has a bound vertex adjacent to it, so that the bound vertex can be dominated.
- If v is required then u is marked as free vertex.

After this labelling the vertex v is deleted from graph and the degree of vertices v and u are updated and reduced. The edge (v, u) is deleted. If new degree of u is 1, then the vertex is added to the queue of end vertices, so that it can be considered later. This process is repeated till all the vertices of tree part are considered and minimum dominating set pertaining to tree part is obtained. By the end of this method all the vertices of tree part would be deleted and we are left with only a cycle, for which we need to find the dominating vertices such that the unicyclic graph has minimum dominating set.

3.2. Method for cycle part:

At this stage the vertices of the cycle might have been already marked as free, bound or required. Observe that if all the vertices of the cycle are already marked as free, then there is no need to find the dominating vertices for tree as all the vertices have been already dominated. If all the vertices of the cycle are marked as bound then it means that all the vertices must be dominated.

In this case we can consider the minimum dominating number as $C_k/3$ [4]. In case the cycle has any mixed combination of marking, then we have to find the minimum dominating set for the cycle separately.

To find minimum dominating set in this case we have to find an edge and remove it from cycle so that it results into a tree. Then by applying the method used for tree part, we can find the minimum dominating set for cycle part.

Selecting an edge to remove from cycle is not a straight forward method, as we can not randomly select an edge and delete. To select this edge the following observations must be used.

- First we have to check if there are any vertices in the cycle marked as required. Then the vertex marked as required has to be considered as start vertex v (the first encountered required is considered as start vertex). Find the adjacent vertex u and delete the edge (v, u) . Update the degree of v and u and add the vertices v and u to the queue of end vertices.
- In case the cycle is not marked with any required vertex then we have to look for free vertex and if a free vertex found then have to consider it as start vertex v . Find the adjacent vertex u . delete the edge (v, u) . Update the degree of v and u . Update the degree of v and u and add the vertices v and u to the queue of end vertices.
- If any of the vertices is not marked as free or required then all the vertices are bound vertices. Then minimum domination number of cycle is given by $C_k/3$. But in algorithm we have to start by the first vertex of the cycle that is encountered.

Finally listing out all vertices whose status is marked as required gives the minimum dominating set and its cardinality gives the domination number for the graph.

Illustration:

Following table has an example for illustration of algorithm on the graph G.

Deleted End vertex and edge	List of end vertices	Vertex Marking	Illustration
-	1, 6, 7, 8, 16, 18, 19	All vertices are bound vertices initially. If marking is not mentioned, then it can be assumed to be bound.
1 (1,4)	6, 7, 8, 16, 18, 19	1-f, 4-r

6 (6, 4)	7, 8, 16, 18, 19, 4	1-f, 4-r, 6-f
7 (7, 10)	8, 16, 18, 19, 4	1-f, 4-r, 6-f, 7-f, 10-r
8 (8, 9)	16, 18, 19, 4	1-f, 4-r, 6-f, 7-f, 8-f, 9-r, 10-r
16 (16, 14)	18, 19, 4	1-f, 4-r, 6-f, 7-f, 8-f, 9-r, 10-r, 14-r
18 (18, 15)	19, 4	1-f, 4-r, 6-f, 7-f, 8-f, 9-r, 10-r, 14-r, 18-f, 15-r
19 (19, 15)	4, 15	1-f, 4-r, 6-f, 7-f, 8-f, 9-r, 10-r, 14-r, 18-f, 15-r, 19-f
4 (4, 3)	15, 3	1-f, 3-f, 4-r, 6-f, 7-f, 8-f, 9-r, 10-r, 14-r, 18-f, 15-r, 19-f

15 (15, 17)	3	1-f, 3-f, 4-r, 6-f, 7-f, 8-f, 9-r, 10-r, 14-r, 15-r, 17-f, 18-f, 19-f
3 (3, 2)	2	1-f, 3-f, 4-r, 5-r, 6-f, 7-f, 8-f, 9-r, 10-r, 14-r, 15-r, 17-f, 18-f, 19-f
2 (2, 5)	-	1-f, 3-f, 4-r, 5-r, 6-f, 7-f, 8-f, 9-r, 10-r, 14-r, 15-r, 17-f, 18-f, 19-f
- (5, 9)	5, 9	(5 is taken as start vertex) 5-r, 9-r, 10-r, 14-r, 17-f
5 (5, 10)	9, 10	5-r, 9-r, 10-r, 14-r, 17-f
9 (9, 17)	10, 17	5-r, 9-r, 10-r, 14-r, 17-f
10 (10, 12)	17, 12	5-r, 9-r, 10-r, 14-r, 17-f, 12-f
17 (17, 14)	12, 14	5-r, 9-r, 10-r, 14-r, 17-f, 12-f
12 (12, 11)	14, 11	5-r, 9-r, 10-r, 14-r, 17-f, 12-f

14 (14, 13)	11, 13	5-r, 9-r, 10-r, 13-f, 14-r, 17-f, 12-f
11 (11, 13)	13	5-r, 9-r, 10-r, 13-r, 14-r, 17-f, 12-f
-	Degree of vertex 13 is 0, but 11 is still not dominated, so 13 is marked as required.		
The minimum dominating set consists of all the vertices marked as required i.e. {4, 5 , 9, 10, 13, 14, 15}			

4. ALGORITHM:

The pseudocode (algorithm) for the above method is given here. Its time performance is linear i.e., $O(n)$. However since we are using matrix for storing the graph we can say it is $O(n^2)$. The variable description is as below.

status[]- An array indicating whether the vertex is free, bound or required.

u and v are the vertices.

list[]- An array holding all the end vertices.

k- The number of end vertices in the list

f=1, Free vertices

b=2, Bound Vertices

r=3, Required Vertices

visited[]- An array indicating whether a vertex is already visited or not. 0 for unvisited and 1 for visited.

4.1. Algorithm for tree part:

The algorithm marks the dominating vertices of the tree part. Before applying this algorithm we have to find the list of end vertices and store them in the array.

ALGORITHM MDomsetTree()

```

{
    for( j=1 to k)
    {
        Let v=list[j], i.e. the end vertex from the list.

        if(status[v] = b)
        {
            Find u which is adjacent of v. Mark v as visited.
            Update the degree of v, by decrementing it by 1.
            if(u != 0) // if u=0 means there are no adjacent vertices.
                //Otherwise we consider the vertex number.

```

```

{
    // Mark u as required Update the degree of u, by
    // decrementing it by 1.

    status[u] = r;

    if( deg[u] = 1 ) // Update end vertex list.
    {
        list[k] = u;
        k++;
    }
}
}
else if( status[v] = f)
{
    Mark v as visited.
    Find u which is adjacent of v.
    Update the degree of v, by decrementing it by 1.
    if (u != 0)
    {
        Update the degree of u, by decrementing it by 1.

        //If u is the last vertex of the tree with degree 0, then it
        //cannot be added to the list of end vertices. Mark it
        //required.
        if( deg[u] = 0 && status[u] = b)
            status[u] = r;

        if( deg[u] = 1)
        {
            list[k] = u;
            k++;
        }
    }
}
}

```

```

else if(status[v]=r)
{
    Mark v as visited.
    Find u which is adjacent of v.
    Update the degree of v, by decrementing it by 1.
    if( u != 0)
    {
        if( status[u] = b) { status[u] = f; }
        Update the degree of u, by decrementing it by 1.

        if(deg[u] = 1)
        {
            list[k] = u;
            k++;
        }
    }
}
}
}

```

4.2. Algorithm for cycle part:

The algorithm finds the start vertex of the cycle part. By using start vertex we have to find the proper edge and remove it.

ALGORITHM MDomsetCycle()

```

{
    initialize k=1

    //Find the start vertex st.
    for(i=1 to n)
    {
        // Find the a required vertex
        if( deg[i] = 2 && status[i] = r)
        {
            st = i;
            flag = 1;

```



```

        break;

    }
    else if( deg[i] = 2 && status[i] = f)//Free vertex
    {
        st=i;
        flag=1;
        break;
    }
}
//If all the vertices of cycle are bound
if(flag != 1)
{
    for(i=1 to n)
    {
        if(deg[i] = 2)
        {
            st = i;
            break;
        }
    }
}

```

Find u which is adjacent of st .

```
if( status[st] = r && status[u] != r)
```

```
{
```

```
    status[u] = f;
```

```
}
```

Mark st as visited.

Mark u as visited.

Update the degree of st , by decrementing it by 1.

Update the degree of u , by decrementing it by 1.

//add the vertex st to the list of end vertices.

```
list[k] = st;
```

```
k++;
```

```

        //add the vertex u to the list of end vertices.
        list[k] = u;
        k++;
    }

```

4.3. Algorithm for Finding the dominating set and domination number of the unicyclic graph:

Finally we can find the minimum dominating set and domination number as below. Here the vertices of the dominating set are displayed. The algorithm can be modified to store them in another array.

Algorithm FindMinDomSet ()

```

{
    // DomNum- Domination Number
    DomNum=0;
    for(i =1 to n)
    {
        if( status[i] = r)
        {
            Display( i );
            DomNum++;
        }
    }
    Display( DomNum );
}

```

5. CONCLUSIONS:

An algorithmic approach is presented to find the domination number and minimum dominating set for the unicyclic graphs. Even though it is NP Complete Problem for general graphs, the domination number can be computed for some class of graphs like trees, unicyclic graphs etc. The details regarding domination and different class of graphs can be seen in [1] and [6]. The algorithm is implemented in C programming (Turbo C compiler). Designing and analysing of algorithms can be referred in [2].

ACKNOWLEDGEMENTS:

I would like to thank Prof. H. B. Walikar, for suggesting this concept. I also thank Prof. S. A. Choudum, former chair of the department of mathematics at IIT Madras, specializing in graph theory. He discussed the algorithm in [3], and gave me clear idea about it.

REFERENCES:

- [1] Andreas Brandstadt, Van Bang Le, Jeremy P. Spinrad, Graph Classes A Survey, SIAM, 1999.
- [2] Cormen. T. H, Leiserson, C. E. Rivest R. L, and C. Stein, Introduction to Algorithms, 2nd Ed. MIT press, Cambridge, MA, 2001.
- [3] E. Cockayne, S. Goodman and S. Hedetniemi, A linear algorithm for the domination number of a tree, Information Processing Letters, Volume 4, Number 2, 1975.
- [4] H. B. Walikar, B. D. Acharya and E. Sampathkumar, Recent developments in the theory of domination in graphs, Mehta Research Institute, Allahabad, MRI Lecture notes in Math., 1, 1979.
- [5] Harary F., Graph Theory, Addison- Wesley, Mass. 1969.
- [6] T. W. Haynes, S.T. Hedetniemi and P.J. Slater, Fundamentals of domination in Graphs, Marcel Dekker, New York (1998).