Global Journal of Pure and Applied Mathematics.
ISSN 0973-1768 Volume 10, Number 2 (2014), pp. 177-186

http://www.ripublication.com

Approximation Algorithm for Total Dominating Set using the Method of Search of Dynamic Maximum Weighted Edge in General Graphs with Improved Performance Ratio

T.N.Janakiraman[a, 1] and Lakshmi Prabha S[b]

[a]Department of Mathematics, National Institute of Technology, Trichy-620015, Tamil Nadu, India. Email: janaki@nitt.edu, Phone:+919894794198.
[b]Department of Mathematics, National Institute of Technology, Trichy-620015, Tamil Nadu, India. Email: jaislp111@gmail.com, Phone:+919865390343.
1 Corresponding Author-T.N.Janakiraman.

Abstract

In this paper, an algorithm is given which is approximation, centralized, greedy and one-phase, to find a minimal total dominating set using the novel method of search of dynamic maximum weighted edge, with worst case time complexity $O(nm^2)$ or $O(n^3)$, where n and m are respectively the number of vertices and number of edges of the graph G. Existing algorithms by others have given the performance ratio to find the total dominating set of a general graph up to $1.5 + \ln(\Delta - 0.5)$. In this paper, the performance ratio of the proposed algorithm is proved to be at most $1 + \ln \Delta$ for finding the total dominating set in general graphs, where Δ is the maximum degree of G.

Keywords. Graph algorithms; approximation algorithms; total dominating set; performance ratio; time complexity.

2010 MSC: 05C85; 05C69

Abbreviations: TDS-Total Dominating Set.

1. Introduction

Some basic definitions from Slater et al. [1] are given below.

Basic Definitions:

The graphs considered in this paper are finite, simple, connected and undirected. For a graph G, let $V(G)$ and $E(G)$ denote its vertex (node) set and edge set respectively and

n and *m* denote the cardinality of those sets respectively. The degree of a vertex *v* in a graph *G* is denoted by $deg_G(v)$. The maximum degree of the graph *G* is denoted by $\Delta(G)$. The length of any shortest path between any two vertices *u* and *v* of a connected graph *G* is called the distance between *u* and *v* and is denoted by $d_G(u, v)$. If there is no confusion, we simply use the notions $deg(v)$ and $d(u, v)$ to denote degree and distance respectively for the concerned graph. For $v \in V(G)$, neighbors of *v* are the vertices adjacent to *v* in *G*. The neighborhood $N_G(v)$ of *v* is the set of all neighbors of *v* in *G*. It is also denoted by $N_1(v)$ or simply $N(v)$. For an edge $e = uv$, *u* and *v* are said to be the end vertices of the edge *e*. An edge *e* is said to be adjacent to an edge *f*, if they have a common vertex.

We say that *H* is a sub graph of a graph *G*, denoted by $H < G$, if $V(H) \subseteq V(G)$ and $uv \in E(H)$ implies $uv \in E(G)$. If a sub graph *H* satisfies the added property that for every pair *u*, *v* of vertices, $uv \in E(H)$ if and only if $uv \in E(G)$, then *H* is called an induced sub graph of *G*. The induced sub graph *H* of *G* with $S = V(H)$ is called the sub graph induced by *S* and is denoted by $<S|G>$ or simply $<S>$.

The concept of domination was introduced by Ore [2]. A set $D \subseteq V(G)$ is called a *dominating set* if every vertex *v* in *V* is either an element of *D* or is adjacent to an element of *D*. A dominating set *D* is a minimal dominating set if $D-\{v\}$ is not a dominating set for any $v \in D$. The domination number $\gamma(G)$ of a graph *G* equals the minimum cardinality of a dominating set in *G*. The readers are also directed to refer Slater et al. [1] for further details of basic definitions, not given in this paper.

Total domination in graphs was introduced by Cockayne, Dawes, and Hedetniemi [3] and is now well studied in graph theory. The literature on this subject has been surveyed and detailed in the two books by Haynes, Hedetniemi, and Slater [1, 4].

A dominating set $D \subseteq V$ is a *total dominating set*, abbreviated TDS, of *G* if every vertex in *V* is adjacent to a vertex in *D*. Every graph without isolated vertices has a TDS, since $D = V$ is such a set. A total dominating set *D* is a minimal total dominating set if $D-\{v\}$ is not a total dominating set for any $v \in D$. The total domination number of *G* denoted by $\gamma_t(G)$ is the minimum cardinality of a TDS of *G*. We use T^* to denote the minimum TDS for *G* (in theorems of this paper, this notation is used). This implies that $\gamma_t(G) = |T^*|$.

The rest of the paper is organized as follows: Section 2 discusses about prior work. Section 3 contains an algorithm and some theorems for proving the correctness of that algorithm. Section 4 compares the algorithm with the other algorithms given by various authors and discusses the performance ratio. Section 5 concludes the paper.

2. Prior Work

NP-completeness of Min-TDS (minimum total dominating set) is discussed in Slater et al. [1, 4]. A survey work has been done related to the results on total domination number of graphs by Henning [5]. The readers are also directed to refer Slater et al. [1, 4] for details of TDS not given in this paper. An approximation, greedy algorithm is given by Zhu [6] for finding the total dominating set with the performance ratio 1.5 + ln (Δ - 0.5). This problem also arises in a number of distributed network applications, where the problem is to locate the smallest number of centers in the

networks such that every vertex is adjacent to at least one of the centers, and their center should also be adjacent to at least one of the other centers for its fault tolerance.

The purpose of this paper is to give an approximation algorithm, using a new approach as well as with better performance ratio for finding TDS in general graphs.

3. Main Results

With each edge, we associate a color namely white, gray and black. With each vertex also, we associate a color namely white, gray and black. Initially all the edges and vertices are white in color. As the algorithm progresses, their colors keep on changing.

3.1. Definitions for the Algorithm:

Definition 1. White Degree of a vertex: A vertex u is called white neighbor of a vertex v, if $u \in N(v)$ and u is white. White degree of a vertex is equal to the number of white neighbors of that vertex. White degree of the vertex u is denoted by $WD(u)$, which is the number of white neighbors of u.

Definition 2. Common White Degree of two Vertices: A vertex w is called a common neighbor of two vertices u and v, if w is adjacent to both u and v. The vertex w is called *common white neighbor* of u and v, if w is white and $w \in N(u) \cap N(v)$. **Common White Degree** of two vertices u and v is equal to the number of common white neighbors of u and v and it is denoted by $CWD(u, v)$.

Definition 3. An edge is said to be a **white edge** if the color of that edge is white. Similarly **black edges** and **gray edges** can be defined. A vertex or edge is said to be non-black, if it is not black in color, that is, it may be gray or white in color.

3.2. Notation:

- $A(G) \rightarrow$ Adjacency matrix of G.
- $D \rightarrow$ The instant vertex subset of $V(G)$ (instant set-the set at a particular iteration) consisting of black **vertices** chosen at that iteration using some processing. Initially, this set is empty. Finally this is the output TDS.
- $E_1 \rightarrow$ The instant edge subset of $E(G)$ consisting of black **edges** chosen at that iteration using some processing. Initially this set is empty.
- $T \rightarrow$ The instant sub graph (graph at a particular iteration) whose vertex set is D and edge set is E_1. It may be connected or disconnected. T may be a tree or cycle at any iteration, that is, $<D>$ may be a tree or cycle.
- $w \rightarrow$ The weight function which associates a natural number to each edge of $E(G)$. For each edge $e = uv$, the weight function is defined as $w(e) = WD(u) + WD(v) - CWD(u, v)$ in run-time, that is, the weight function is calculated at each iteration of the algorithm.
- Performance Ratio $\rightarrow |D| / |T^*|$ (in this paper).
- In general, Performance Ratio = |minimal set| / |minimum set|, where minimal and minimum set represent the minimal parameter.

3.3. Outline of the Algorithm:

The proposed algorithm finds a TDS in a graph in which edges are weighted dynamically and *maximum* weighted edges are searched to obtain better optimal TDS. The new approach in this algorithm is that the edge weight is assigned dynamically as the sum of the *white degrees* of the end vertices of that edge minus common white neighbors of those end vertices of the same edge, in the run-time and then the edges are chosen randomly based on the dynamic maximum weight.

The algorithm is based on coloring mechanism.

1. Initially, all the edges and vertices are colored white.
2. Calculate the white degree of each vertex (gray and white) of $V(G)$.
3. For each edge (gray and white), the edge weight w is assigned as the *sum of white degrees of the end vertices* of that edge minus common white neighbors of those end vertices of the same edge.
4. Sort the edges into non-increasing order by the weight w.
5. An edge (gray or white) of maximum weight, is chosen. Then that edge together with its end vertices are colored black and are added to the instant sub graph T;
6. The adjacent white edges of that black edge (chosen in the previous step), are colored gray, and their non-black end vertices are also colored gray.
7. This whole process from 2 to 6 is continued until there exists no white vertex.
8. **The set of all black edges (black vertices) forms TDS.**

3.4. Approximation Algorithm for finding TDS of a given graph

The single phase approximation algorithm for finding TDS is as follows:

Input: The graph G (the adjacency matrix $A(G)$).

Output: D, which is the TDS.

Algorithm:

TDSDMWE($A(G)$)

1. Color all the edges of $E(G)$ and the vertices of $V(G)$ white.
   Initialize $D \leftarrow \emptyset$ and $E_1 \leftarrow \emptyset$.
2. **while** there exists a white vertex
   {
   a) **for** each non-black vertex $x \in V(G)$, calculate $WD(x)$.
   b) **for** each edge $e = uv$ of $E(G) - E_1$, assign the edge weight as $w(e) = WD(u) + WD(v) - CWD(u, v)$.
   c) Sort the edges of $E(G) - E_1$ into non increasing order by weight w.
   d) Choose an (gray or white) edge $e = uv$ from $E(G) - E_1$ of *maximum weight* w. // E_1 is subtracted from $E(G)$ to avoid revisiting those edges of E_1.
   e) Color e black and all its adjacent white edges gray. Color the vertices u and v black and all the non-black adjacent vertices of u and v, gray.
   f) $D \leftarrow D \cup \{u, v\}$ and $E_1 \leftarrow E_1 \cup \{e\}$.
   }
3. Print D and stop.

Note 1. Black edges have end vertices which will be only black vertices; gray edges have end vertices which may be black or gray (or both) vertices; the white edges have

end vertices which may be gray or white (or both) vertices.

Note 2. In the above algorithm, ties in edge weights can be broken arbitrarily. But a special case has to be noted. Consider Figure 1.

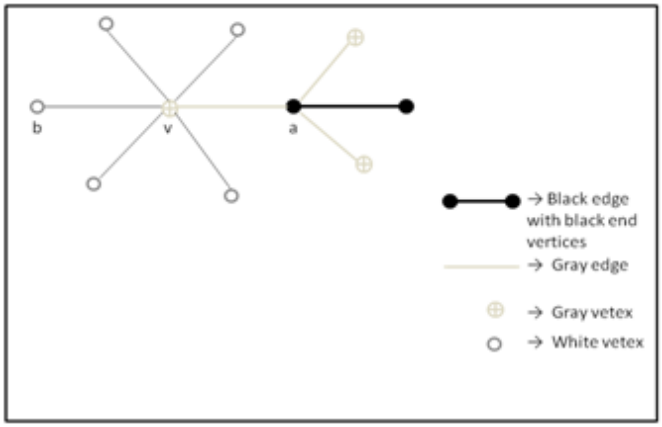

Figure 1: Special Case

Suppose a gray edge *av* and a white edge *vb* are adjacent and have same edge weight, say w_1. It means that *v* is a gray vertex. Suppose $WD(v) = w_1$ and $WD(b)= 0$. If the gray edge is chosen to be added to the instant sub graph *T*, then *v* alone is added to *T* (as *a* is already in *T*). If the white edge is chosen to be added to *T*, then two vertices *b* and *v* have to be added to *T*, thus leading to an increase in $|D|$. Hence in this case, tie can be broken by choosing the gray edge than the white edge. Refer Figure 1.

3.5. Illustration for the Algorithm:

Let us consider the graph *G* as in Figure 2.

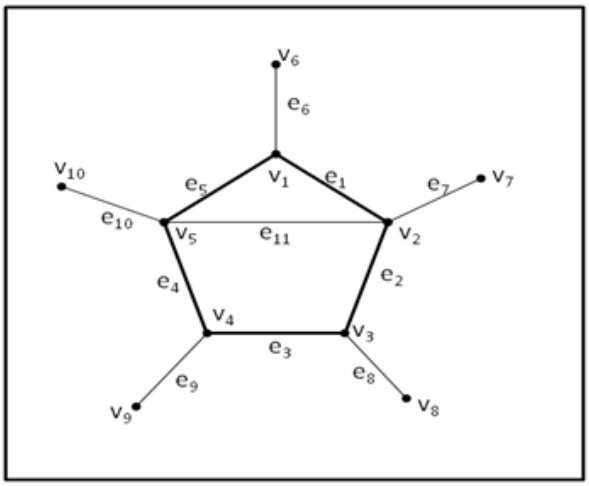

Figure 2: Illustration Figure

Initially, the edge weights are $w(e_2) = w(e_4) = w(e_{11}) = 7$, $w(e_1) = w(e_3) = w(e_5) = 6$, $w(e_7) = w(e_{10}) = 5$ and $w(e_6) = w(e_8) = w(e_9) = 4$.

In the first iteration, let us choose the edge e_4. Then the edge e_4 and its end vertices are colored black; the edges e_5, e_{11}, e_3, e_9 and e_{10} are colored gray and their end vertices, v_1, v_2, v_3, v_9 and v_{10} are colored gray.

In the second iteration, $w(e_1) = w(e_2) = 2$, $w(e_3) = w(e_5) = w(e_6) = w(e_7) = w(e_8) = w(e_{11}) = 1$, and $w(e_9) = w(e_{10}) = 0$. Let us choose e_2, the edge together with its end vertices are colored black; the edges e_1, e_7 and e_8, together with their end vertices are colored gray.

In the third iteration, $w(e_1) = w(e_5) = w(e_6) = 1$ and the remaining edges have weight 0. By Note 2, we can choose either e_1 or e_5. Let us choose e_1. The edge together with its end vertices are colored black; The edge e_6 is colored gray and its end vertices are colored gray. Now, there exists no white vertex in G.

Thus, $E_1 = E(T) = \{e_4, e_2, e_1\}$, $D = V(T) = \{v_4, v_5, v_2, v_3, v_1\}$ and $|D| = 5$, which is the cardinality of minimal TDS of G.

3.6. Correctness of the Algorithm TDSDMWE

The following theorem gives the correctness of the algorithm for TDS.

Theorem 1. *D is a minimal total dominating set.*

Proof. D consists of the set of all black vertices of T. The vertices of D either dominate $V(G)$ or avoid isolated vertices in $<D>$. Among the end vertices of the edge, chosen to be added to D, if it is a white vertex with at least one white neighbor, then it is a dominating vertex. If it is a white vertex with gray neighbor, then either it is chosen for dominating itself or for avoiding isolated vertices in $<D>$. As in each iteration, edges are chosen, no vertex is isolated in $<D>$. The above discussion implies that $D-\{v\}$, for any $v \in V(G)$, is not TDS for G. Thus, D forms a minimal total dominating set for G. □

Next, we compute the worst case time complexity of the algorithm.

3.7. Time Complexity:

The running time of the algorithm TDSDMWE depends on step 2 of the algorithm. In Step 2, at **each iteration of the algorithm**, we execute two processes mainly on which the time complexity is dependent. We

(i) Calculate the white degree of each vertex of the graph and

(ii) Sort the edges (in non-increasing order).

In one iteration, for calculating the white degree of each vertex of G, we visit every element in $A(G)$. In worst case, finding $A(G)$ takes $O(n^2)$, so calculating the white degree of each vertex will take $O(n^2)$. Sorting the edges takes $O(m \lg m)$ in the worst case, where lg denotes binary logarithm. As $m \lg m < m^2$, let us assume that sorting takes $O(m^2)$ in the worst case.

As these two processes are executed until there exists no white vertex in G, these two steps will be executed at most $|V(G)| = n$ times in the worst case. Hence the time

taken for the process (i), for all iterations, in the worst case is $O(n^3)$ and the time taken for the process (ii), for all iterations, in the worst case is $O(nm^2)$. Thus, the worst case time complexity of the algorithm is either $O(n^3)$, if $n > m$ or $O(nm^2)$, if $m > n$, where n is the number of vertices in G and m is the number of edges in G.

4. Comparison with the Other Algorithms

The advantage of our algorithm is that at each iteration, the edge is chosen based on dynamic maximum weight. Thus, at each iteration, large numbers of white vertices are absorbed, that is, large numbers of vertices are dominated at each iteration, thus paving way for the *faster convergence* towards the solution.

Next theorem provides the performance ratio of the algorithm for finding TDS.

Theorem 2. *For any graph G, the performance ratio of the Algorithm TDSDMWE for finding TDS is at most 1 + ln Δ.*

Proof. A piece is defined as either a white vertex or a black connected component (a black component is a connected component formed by black vertices and gray vertices). An iteration is defined as executing the while statement (step 2) once in the algorithm TDSDMWE. Let a_0 denote the number of pieces left after 0^{th} iteration (initial stage). Then, $a_0 = n$, where n is the number of vertices in G. It is to be noted that, at any intermediate iteration, there will be at least one black component and at least one white vertex. **If there is no white vertex, it means that the process is completed** and there will be only one black component. Let a_i denote the number of pieces left after the i^{th} iteration. Hence by the definition of piece and a_i, $\boldsymbol{a_i \geq 2}$ **until there exists at least one white vertex**. Also,

$$\boldsymbol{a_i = 1} \qquad (1)$$

if all the white vertices are exhausted.

Let k be the total number of iterations. Then, $k \leq |D|$ (because in each iteration, we choose at least one vertex or two vertices and add them to D). Consider the $(i+1)^{th}$ iteration. The optimal solution can connect a_i pieces, that is, $|T^*|$ nodes can connect a_i pieces. Hence the greedy procedure is guaranteed to pick up a node, which connects at least $\lceil a_i / |T^*| \rceil$ components. Thus, the recurrence relation is:
$a_{i+1} \leq a_i - \lceil a_i / |T^*| \rceil \leq a_i - a_i / |T^*|$ (since $\lceil x \rceil \geq x$). **Once $\boldsymbol{a_i}$ reaches 1, the algorithm is stopped** and further a_i s, (that is, a_{i+1}, a_{i+2}, and so on) are assumed to be 1 as there will be only one connected component thereafter, (by equation 1).

This implies that $a_k = a_{k+1} = \ldots = 1$ (since k denotes the total number of iterations) and hence $\boldsymbol{a_{|D|} = 1}$ (since $k \leq |D|$).

Now, we have,

$$a_i \leq a_{i-1} (1 - 1 / |T^*|) \leq \ldots \leq a_0 (1 - 1 / |T^*|)^i \leq n \exp(-i / |T^*|) \qquad (2)$$

since $1+x \leq \exp(x)$, for $x > -1$.

As $a_i \leq a_{i-1} - 1$, we have $a_i \leq a_{i-1} - 1 \leq a_{i-2} - 2 \leq \ldots \leq a_{i-|T^*|} - |T^*|$, which implies, $a_{|D|-|T^*|} \geq |T^*| + 1$ as $a_{|D|} = 1$.

Substituting $i = |D| - |T^*|$ in equation 2, we get,

$|T^*| + 1 \leq n \exp(-i / |T^*|)$,
which implies $i \leq |T^*| \ln (n / (|T^*|+1))$,
which in turn implies, $i \leq |T^*| \ln \Delta$, since $(n / |T^*|) \leq \Delta+1$.
Thus, $|D| = i + |T^*| \leq (1 + \ln \Delta) |T^*|$. □

4.1. Similarity and Difference:

The following remarks provide the similarity and difference in calculation of performance ratio of our algorithm over other algorithms. Here, we compare our algorithm with the algorithms, given by Guha et al. [7] and Du et al. [8], for finding connected dominating set.

Remark 1. Comparison between our algorithm and the algorithm of Du et al. [8]:

Similarity: Proof technique procedure (proof of Theorem 2 in this paper resembles the Theorem 3.4 in [8]).

Difference:

1. Definition of a_i ;
2. Recurrence relation (because of the definition of a_i);
3. Equation differing in our proof and their [8] proof.
   - **Theirs:** $a_{|C\_G|} = 2$ (because of the definition of a_i).
   - **Ours:** $a_{|D|} = 1$, (C_G is D according to our notation).
4. Inequalities differing in our proof and their [8] proof.
   - **Theirs:** As $a_{|C\_G|} = 2$, they got $a_{|C\_G|-2|C^*|} \geq 2|C^*| + 2$.
   - **Ours:** As $a_{|D|} = 1$, we got $a_{|D|-|T^*|} \geq |T^*| + 1$.
   - The other inequalities following the above one in their proof are also different from those in our proof.

Remark 2. Comparison between our algorithm and the algorithm of Guha et al. [7]:

Similarity: Definition of piece.

Difference:

1. Proof technique;
2. Recurrence relation.
   - **Theirs:** $a_i \leq a_{i-1} (1 - 1 / |C^*|) + 1$
   - **Ours:** $a_i \leq a_{i-1} (1 - 1 / |T^*|)$

Explanation:

Their recurrence relation needs '**plus one**'. But, we do not need that 'plus one' factor, because, we color two vertices black in many iterations (this is possible because of our new approach of algorithm) and also as explained in the Theorem 2, "$a_i \geq 2$ until there exists at least one white vertex. Also, $a_i = 1$ once if all the white vertices are exhausted".

5. Conclusion

In this paper, an approximation, centralized, greedy, one-phase algorithm is given using a **new approach** for constructing TDS. The performance ratio of the algorithm for finding TDS is proved to be at most $1 + \ln \Delta$, for general graphs, which is better than the available related algorithms in the literature. Also, the worst case time complexity of the algorithm has been analyzed and proved to be $O(nm^2)$ or $O(n^3)$, where Δ, n and m are respectively, the maximum degree, the number of vertices and the number of edges of the given graph G.

Generally the existing algorithms use the basic common methodology given by Johnson [9], Lovasz [10] and Slavík [11]: "As long as there are uncovered nodes, the greedy algorithm picks up a node which covers the maximum number of uncovered nodes and puts it into the set" to find dominating set (set cover). We have also used it but slightly modified this basic and common methodology in this paper, that our greedy procedure picks up an *edge* which covers the maximum number of uncovered nodes, to find total dominating set. This paved way for **faster convergence** towards the solution.

6. Future Directions

We hope that the performance ratio of the algorithm can be further reduced by using some other methodology. But Guha and Khuller [7] has proved that there does not exist a polynomial-time approximation for finding connected dominating set (applicable for TDS also) with performance ratio $\rho\, H(\Delta)$ for $0 < \rho < 1$ unless NP $\subseteq$ TIME $(n^{O(\log \log n)})$, where Δ is the maximum degree in the input graph. So, the performance ratio cannot be reduced lesser than $\ln \Delta$, but it can be reduced from $1 + \ln \Delta$ to $c + \ln \Delta$, where $0 \leq c < 1$ using some other procedure.

Also, the algorithm is observed to perform much better practically. So, the algorithm can be implemented to find out the minimum possible performance ratio. Any reduction in the performance ratio is useful in the practical situations because the algorithm can be applied to solve practical problems which arise in a number of network applications.

References.

[1] Haynes, T.W., Hedetniemi, S. T., and Slater P. J., 1998, "Fundamentals of domination in graphs", Marcel Dekker, New York.

[2] Ore, O., 1962, "Theory of Graphs", Amer. Soc. Colloq. Publ. vol. 38. Amer. Math. Soc., Providence, RI.

[3] Cockayne, E.J., Dawes, R. M. and Hedetniemi, S. T., 1980, "Total domination in graphs", Networks, 10, 211-219.

[4] Haynes, T.W., Hedetniemi, S. T., and Slater P. J., 1998, "Domination in Graphs: Advanced Topics", Marcel Dekker, Inc. New York.

[5] Henning, M. A., 2009, "A survey of selected recent results on total domination in graphs", Discrete Mathematics, 309, 32-63.

[6] Zhu, J., 2009, "Approximation for Minimum Total Dominating Set", ICIS, Seoul, Korea.
[7] Guha, S. and Khuller, S., 1998, "Approximation algorithms for connected dominating sets", Algorithmica, 20(4), 374-387.
[8] Du, H., Jia, X., Ko, K-I, Li, Y., Ruan, L., and Wu, W., 2004, "A greedy approximation for minimum connected dominating sets", Theoretical Computer Science, 329(1-3), 325-330.
[9] Johnson, D. S., 1974, "Approximation Algorithms for Combinatorial Problems", Journal of Computer and System Sciences, 9, 256-278.
[10] Lovasz, L., 1975, "On the Ratio of Optimal Integral and Fractional Covers", Discrete Mathematics, 13, 383-390.
[11] Slavík, P., 1996, "A Tight Analysis of the Greedy Algorithm for Set Cover", In Proc. of the 28th ACM Symposium on Theory of Computing (STOC), 435-441.

Biography:

T.N. Janakiraman is currently a Professor of Department of Mathematics, National Institute of Technology, Trichy, Tamil Nadu, India. He completed his undergraduate Studies at Madras University, India in 1980 and completed his Post graduation at National College, Trichy, India in 1983. He did his Ph.D. in Mathematics (Graph Theory and its applications) at Madras University with a UGC sponsored research fellowship and received his doctoral degree in the year 1991. He was a Postdoctoral Research associate for 1 year (1993-1994) in Madras University. He has two sponsored research projects to his credit and published around 70 papers in refereed National/International journals. His research interests include Pure Graph Theory, Applications of Graph Theory to Fault tolerant networks, Central location problems, Clustering of wired & wireless ad hoc networks, Clustering of cellular and flexible manufacturing models, Image processing, Psychology, Social Networks, Graph coding and Graph Algorithms.

Lakshmi Prabha S is currently a research scholar of Department of Mathematics, National Institute of Technology, Trichy, Tamil Nadu, India. She received her B.Sc., degree in 2008 and M.Sc., degree in 2010 in Mathematics from Alagappa University, Karaikudi, Tamil Nadu, India. Her current research interests include Pure Graph Theory, Graph theory and its applications to Psychology, Social Networks, and graph algorithms.