A Classical Approach for Generation of Minimum Spanning Tree

[1]Sanjay Kumar Pal [2]Samar Sen Sarma

Abstract

This paper intends to introduce some innovative concepts for generation of Minimum Spanning Tree (MST) of weighted graph G. We will also formulate a new circuit testing algorithm to find out circuits during generation of Minimum Spanning Tree of the graph, along with an attempt to optimize the execution time of the algorithms. The primary aim of this paper is to represent the minimum spanning tree algorithms in a simple and efficient manner.

Keywords: Minimum Spanning Tree, Algorithm, Circuit, Tree.

1. Introduction

A Spanning Tree is a tree of a connected graph G, which touches all vertices of the graph. It has wide applications in computer science, electrical engineering, network design and many others fields of engineering and applications. In this regard, the generation of all spanning trees of a graph G [1, 5, 6, 7, 8, 9, 10, 11, 12, 13], is a critical as well as crucial task and it has wide applications in network design and mobile computing. The generation of minimum spanning tree of a graph G is historic and has many significant applications in the field of science, engineering, networking and operations research etc [2, 3, 4, 17].

[1]Lecturer, Department of Computer Science & Applications, NSHM College of Management & Technology, Kolkata – 700 053, INDIA. Email : pal.sanjaykumar@gmail.com

[2]Department of CSE, University of Calcutta, Kolkata. Email : sssarma2001@yahoo.com

Many practical applications, particularly in communication networks and transportation networks can be formulated as optimization of minimum spanning tree problem [2, 3, 4, 14, 15, 16]. The goal in optimization of minimum spanning tree is to find a solution that is appropriate for a particular application. When studying problems of spanning trees, one often makes an assumption of a general position for minimum spanning trees, he infinitesimally perturbs the edge weights so that all are distinct, thus, picking out a unique solution. There are several algorithms which exist for generation of Minimum Spanning Tree. Otakar Boruvka described an algorithm for finding a Minimum Spanning Tree in a graph for which all the edge weights are distinct [2]. In 1957, C. Prim, the renowned computer scientist, discovered another algorithm that finds a minimum spanning tree for a connected weighted graph [2, 3, 4]. This algorithm continuously increases the size of a tree starting with a single vertex until it spans all the vertices. This algorithm was actually discovered by mathematician Vojtech Jarnik in 1930. Joseph Kruskal described a minimum spanning tree algorithm where total weight of all the edges in the tree is minimized [2, 3, 4]. Edsger Dijkstra in the year 1959 discovered a minimum spanning tree algorithm that solved the single source shortest path problem for a directed graph with non-negative edge weights [2, 3, 4, 17]. Here, in this paper we are going to describe and analyse some minimum spanning tree algorithms with innovate ideas. Here, in this paper we are going to apply new algorithm for testing of circuits in the way of generation of Minimum Spanning Tree and obtaining better result.

2. Definitions

An undirected, simple, connected graph G is an ordered triple (V(G), E(G), f) consist of (i) a non empty set of vertices $n \in V$ of the graph G (ii) a set of edges $e \in E$ of graph G and (iii) a mapping f from the set of edges E to a set of unordered pair of elements of V.

2.1 Spanning Subgraph

Spanning subgraph is a subgraph of a graph G containing all the vertices of the graph G.

2.2 Tree

A Tree is a subgraph of a graph G without any circuit or self loop.

2.3 Spanning Tree

A Spanning Tree of a graph G is a spanning subgraph that is itself a tree.

2.4 Minimum Spanning Tree

A Minimum Spanning Tree is a Spanning Tree of a weighted graph with minimum weight of edges.

3. Circuit Tester

3.1 Circuit Property

i) Let T be a minimum spanning tree of a weighted graph G.

ii) Let e be an edge of g that is not in T and let C be the circuit formed by e with minimum spanning tree T.

iii) For every edge h of circuit C, weight(h) ≤ weight(e).

3.2 Proof

i) By contradiction

ii) If weight (h) > weight (e) we can get a spanning tree of smaller weight by replacing e with h.

Theorem1: If a subgraph of n-1 edges contains more than three nodes of degree more than one and if there is no pendent edge in the graph, the subgraph contains a circuit.

Proof: For simplicity and easy to explain the theorem we consider a simple connected graph, shown in figure 1.

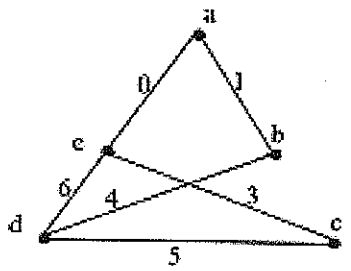

Figure 1 : A Simple Connected Graph

Form the given graph in figure 1, if we consider the edge combination, 1 4 5 6 2, the degree of each vertices corresponding to the given edge combination are,

Node No. :	*a*	*b*	*c*	*d*	*e*
Degree :	1	3	1	3	1

Since there are three vertices of degree one and only two vertices of degree more than one, hence this n-1 edges combination will not produce a tree of the graph G. The pictorial form of this tree is shown in figure 2.

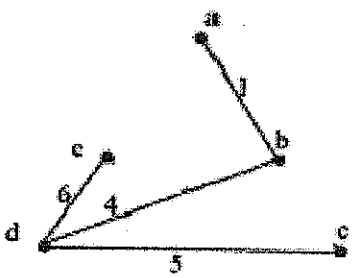

Figure 2 : An Illustrative Tree of Graph in Figure1

Considering another example, if edge combination is, 0 3 5 6 2, of the graph shown in figure 1, the degree of each vertices corresponding to the given edges combination are,

Node No. :	*a*	*b*	*c*	*d*	*e*
Degree :	1	1	2	2	3

In the above combination two vertices of degree one and three vertices of degree more than one, hence this

combination may give the circuit. Deleting pendent edges incidence on vertex a and b, the modified degree of all the vertices are,

Node No. :	*a*	*b*	*c*	*d*	*e*
Degree :	0	0	2	2	2

Since, the degree of all the three vertices are more than one, this is confirm that the edge combination will produce a circuit. The pictorial form of this combination is shown in figure 3.

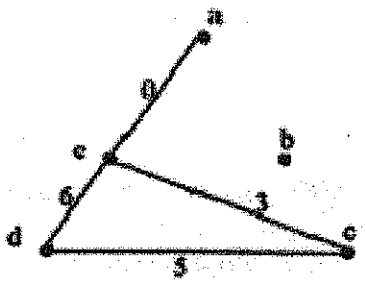

Figure 3 : An Illustrative Circuit of Graph in Figure 1

3.3 Description of Circuit Testing Algorithm

The Circuit Generator is an implementation of algorithm for finding out a set of circuits of a finite undirected graph from its adjacency matrix.

The input parameters to the Circuit Testing method are :

i) A modified form of adjacency matrix, called A.

i) The number of vertices of the graph, called n.

ii) The number of edges of the graph, called e

The output show the combination is circuit or not.

4. New Circuits Testing Algorithm

Here we introduce a new circuit testing algorithm with the existing minimum spanning tree generation algorithms to find out the cycle in newly constructed tree of the graph.

This algorithm ascertains us whether edge combinations of minimum spanning tree form a circuit [2, 3, 4] or not.

The circuit testing technique using new circuit testing algorithm already discussed with example in theorem 1.

Step 1: From the combination of edges and incidence matrix, obtain degree of each node contributed by the edges under consideration.

Step 2: Test whether at least two nodes of degree one? If not, go to step 6. Otherwise continue.

Step 3: Test whether at least three nodes of degree more than one? If not go to step 5.

Step 4: Delete pendant edges, if exists of n-1 edges and modify the degree of the nodes accordingly and go to step 2. Otherwise go to step 6.

Step 5: Edge combinations are tree.

Step 6: Stop.

5. New Approach Of Mst Generation Algorithms

This section describes the three famous minimum spanning tree generation algorithms of Boruvka, Prim-Jarnik, and Kruskal in innovative way. We introduced new circuit testing to find out the minimum spanning of the graph using the existing algorithms. The reason behind this is to minimize the execution time of program of the existing algorithm. The new circuit algorithm is given in section 4 and explained with example inside the theorem 1.

5.1.1 Boruvka's Algorithm

Boruvka's Algorithm begins with examining each vertex and adding the edges of minimum weight from that vertex to another vertex in the graph [2], except those edges already added. This process will be continued until all the vertices are not included in the minimum spanning tree.

5.1.2 Pseudocode of Boruvka's Algorithm

- Begin with a connected graph G containing edges of distinct weight, and an empty set of edges T

- While the vertices of G connected by T are disjoint
  - Begin with an empty set of edges E
  - For each component
    - Begins with an empty set of edges S
    - For each vertex in the component
      - Add the edge of minimum weight from the vertex in the component to another vertex in a disjoint component to S
    - Add the minimum weight edge in S to E
  - Add the resulting set of edges E to T.
- The resulting set of edges T is the minimum spanning tree of G.

5.1.3 Algorithm of Boruvka

1. Algorithm BoruvkaMST(G)
2. $T \leftarrow V$ // the vertices of graph G
3. while T has fewer than n-1 edges do
4. for each connected component C in T do
5. let edge e be the minimum weight edge from C to another component in T
6. if e is not already in C then
7. add edge e to T
8. return T

5.2.1 Prim-Jarnik Algorithm

Prim-Jarnik algorithm finds a minimum spanning tree for a connected weighted graph [2, 3, 4, 16]. It finds a subset of edges that form a tree and includes every vertex of the graph. This algorithm continuously increases the size of the tree starting with a single vertex until it spans all the vertices.

5.2.2 Pseudocode of Prim-Jarnik Algorithm

- Pick an arbitrary vertex v_a and grow minimum spanning tree starting from v_a.
- Store each vertex v a label d(v) = the smallest weight of an edge connecting v to a vertex in the cloud.
- At each step
  - We add to the cloud the vertex u outside the cloud with the smallest distance label
  - Update the labels of the vertices adjacent to u.
- A primary queue stores the vertices outside the cloud
  - Key: distance
  - Element: vertex
- Locator based methods
  - Insert (k, e) return a locator
  - Replace key (l, k) changes the key of an item.
- Store three labels with each vertex
  - Distance
  - Parent edge in minimum spanning tree

  Locator in priority queue.

5.2.3 Algorithm of Prim-Jarnik

1. Algorithm PrimJarnikMST(G)
2. $Q \leftarrow$ new heap based priority queue
3. $s \leftarrow$ a vertex og G
4. for all $v \in G$, vertices()
5. if v = s
6. setDistance(v, 0)
7. else
8. setDistance(v, ∞)

5.3.1 Kruskal Algorithm

Kruskal's Algorithm finds a minimum spanning tree for a connected weighted graph [2, 3, 4, 16]. It finds a subset of edges that forms a tree that includes every vertex, where total weight of all the edges in the tree is minimized. For non connected graph, it finds a minimum spanning forest.

5.3.2 Pseudocode of Kruskal Algorithm

- A priority queue stores the edges outside the cloud
  - Key: weight
  - Element: edge
- At the end of the algorithm
  - We are left with one cloud that encompasses the Minimum Spanning Tree

- A tree T is our Minimum Spanning Tree.

5.3.2.1 Algorithm of Kruskal

1. Algorithm KruskalMST(G)
2. for each vertex v in G do
3. Define an elementary cloud $C(v) \leftarrow \{v\}$
4. Initialized a priority queue Q to contain all edges in G, using the weight as keys.
5. Define a tree $T \leftarrow \phi$
6. while T has fewer than n-1 edges do
7. $(u,v) \leftarrow Q$.removeMin()
8. Let C(v) be the Cloud containing v, and let C(u) be the Cloud containing u.
9. if $C(v) \neq C(u)$ then
10. Add edge (v, u) to T
11. Merge Cloud C(v) and Cloud C(u) into one Cloud, that is, union of C(v) and C(v)
12. return tree T.

6. Results And Conclusion

Hardware used to carry out this experiment is Pentium IV computer and 512 MB RAM. The program is written in 'C' programming language and Turbo 'C' compiler is used for compilation and execution purpose. The experiment has been performed on different graphs with different number of nodes. The storage requirement of this algorithm is proportional to n^2, where n is the number of vertices in the graph G. The execution time required for each algorithm is given hereunder in table-1 and also shows the chart as comparative study of the execution time three algorithms.

Table 1: Execution time the of Algorithms

No. of Node	No. of Edge	Execution Time of Algorithm * 100 (in Sec.)					
		Kruskal		Prim		Boruvka	
		Existing	New	Existing	New	Existing	New
4	5	1.70	1.71	1.91	1.95	1.76	1.76
6	12	8.92	8.84	9.01	8.90	9.50	8.90
9	24	25.54	24.44	22.32	21.75	25.44	24.71
10	42	53.12	51.40	28.98	27.24	51.01	49.42
12	20	41.46	40.32	41.87	40.02	42.99	42.32
13	70	64.88	63.74	53.79	51.73	59.22	57.68
15	72	103.39	101.50	81.33	78.56	128.02	124.95
16	48	96.77	94.25	83.88	79.35	88.24	85.60
17	41	99.32	96.05	99.33	93.60	117.75	112.35
20	95	320.44	311.60	285.45	231.81	333.28	317.41
23	126	599.64	596.40	327.76	306.63	488.99	475.64

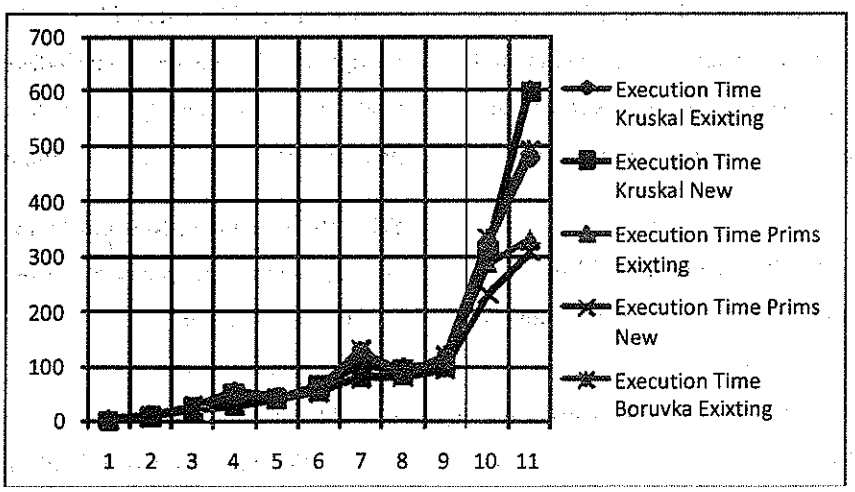

Figure 4: A Comparative Study of the Algorithms

REFERENCES

[1]. A. Rakshit, A. K. Choudhury, S. S. Sarma and K. K. Sen, *"An Efficient Tree Generation Algorithm"*, IETE, Vol. 27, PP. 105-109, 1981.

[2]. N. Deo, *"Graph Theory With Application to Engineering and Computer Sciences"*, PHI, Englewood Cliffs. N. J.

[3]. Thomas H. Coremen et. al., *"Introduction to Algorithms"*, PHI, Second Edition, 2008.

[4]. Harowitz Sahnai & Rajsekaran, *"Fundamentals of Computer Algorithms"*, Galgotia Publications Pvt. Ltd., 2000.

[5]. Sanjay Kumar Pal and Samar Sen Sarma, *"An Efficient All Spanning Tree Generation Algorithm"*, IJCS, Vol. 2, No. 1, PP. 48 – 59, Jan. 2008.

[6]. S. Kapoor and H. Ramesh, *"An Algorithms for Enumerating All Spanning Trees of Undirected and Weighted Graphs"*, SIAM J. Computing, Vol. 24, PP. 247-265, 1995.

[7]. A. Shioura and A. Tamura, *"Efficiently Scanning All Spanning Trees of an Undirected Graph"*, J. of Operation Research Society of Japan, Vol. 38, PP. 331-344, 1995.

[8]. A. Shioura, A. Tamura and T.Uno, *"An optimal Algorithm for Scanning All Spanning Trees of Undirected Graphs"*, to appear in SIAM J. on Computing.

[9]. Kenneth Sorensen and Gerrit K. Janssens, *"An Algorithm to Generate All Spanning Trees of a Graph in Order of Increasing Cost"*, Pesquisa Operacional, Vol. 25, PP. 219- 229, 2005.

[10]. S. Kapoor and H. Ramesh, *"An Algorithm for Enumerating All Spanning Trees of a Directed Graphs"*, Algorithmica, Vol. 27, PP. 120-130, 1997.

[11]. T Matsui, *"A Flexible Algorithm for Generating All Spanning Trees in Undirected Graphs"*, Algorithmica, Vol. 18, PP. 530-543, 1997.

[12]. G. J. Minty, *"A Simple Algorithm for Listing All the Trees of a Graph"*, IEEE Trans. On Circuit Theory, CT-12, PP. 181-185, 1965.

[13]. H. Gabow and E. Myres *"Finding All Spanning Trees of Directed and Undirected Graphs"*, SIAM J. Computing, Vol. 7, PP. 280-287, 1978.

[14]. R.J. Wilson, *"History of Graph Theory"*, Section 1.3, Handbook of Graph Theory, PP. 29 – 49, 2004.

[15]. S. Arunmugam and S. Ramachandran, *"Invitation to Graph Theory"*, Scitech Publication, Chapter – 1, Introduction, Edition: July 2002.

[16]. R.J. Wilson, *Introduction to Graph Theory*, Pearson Education, 2000.

[17]. D. S. Cheema, *"Operations Research"*, Laxmi Publications, Edition: 2006.

Author's Biograpghy

Sanjay Kumar Pal has completed B.Sc. from Calcutta University in 1989, Licentiate Mechanical Engineer from West Bengal State Council for Engineering and Technical Education in 1993 and Master in Computer Application from IGNOU in 2004. Author has more than 20 papers to his credit in the field of Graph Theory, Network Algorithm Design, Software Engineering and Mmanagement published in National and International Journals. He has eight years of experience in engineering industry, five years in software

industry and six years in education. At present he is a Lecturer in Department of Computer Science and Applications, NSHM College of Management and Technology of West Bengal University of Technology, Kolkata

Samar Sen Sarma has completed M.Tech in Radio Physics from Calcutta University and Ph.D. In Computer Science & engineering from Calcutta University. He Has published more than 150 research paper in the field of Digital Electronics, VLSI design, Graph Theory and Algorithm design and in several International and National Journal. At present he is a Professor in the Department of Computer Science and Engineering in The University of Calcutta.