ISSN: 2277-3754
ISO 9001:2008 Certified
International Journal of Engineering and Innovative Technology (IJEIT)
Volume 3, Issue 2, August 2013

K-Edge-Connectivity: a new Approach of finding Minimum Spanning Tree Ordered Minimum Spanning Tree (OMST)

H.B Walikar, Ramesh K, Hanumantu

Abstract - While forming reliable communication networks, we must guarantee is that, after failure of a node or link, the surviving network still allows communication between all other nodes by choosing alternate path which gives strict requirement on the connectivity of the corresponding graph. For a general network design problem it is required that the underlying network to be resilient to link failures is known as the edge-connectivity survivable network design problem. In this paper we present a method called Ordered Minimum Spanning Tree (OMST) used to parallelize efficiently Kruskal's Minimum Spanning Forest algorithm. This algorithm is known for exhibiting inherently sequential characteristics. More specifically, the strict order by which the algorithm checks the edges of a given graph is the main reason behind the lack of explicit parallelism. Our proposed scheme attempts to overcome the imposed restrictions and improve the performance of the algorithm.

***Keywords:-*Minimum Spanning Forest; Parallel algorithms; Ordered Minimum Spanning Tree (OMST), Survivable network;**

I. INTRODUCTION

Connectivity problem is a fundamental property of graph theory [TS92, Wes01], which has important applications in network reliability analysis and network design problems and have many applications in computer science and operation Research. The problem of finding a minimum cost k-connected spanning sub graph of a given a given graph. This is a central algorithmic problem, which is known to be NP-complete when the connectivity requirement is greater than one. Several approximation algorithms have been reported in the literature for this problem. Recently, much effort has been devoted to problems of finding minimum cost sub graphs of a given weighted graph that satisfy given connectivity requirements. A particular important class is the problems with uniform connectivity requirements, where the aim is to find a cheapest spanning sub graph which remains connected in presence of up to k arbitrary edges or vertex failures (i.e., a minimum cost k-edge- or k-vertex connected spanning sub graph, respectively). The widespread adoption of multicore platforms has offered the opportunity to explore new implementation techniques for many algorithms that were initially designed for uniprocessors. By devising new parallel schemes, the programmers will be able to exploit in a more efficient way the multiple hardware contexts offered in today's platforms. A category of problems among the most difficult to parallelize are the ones that exhibit inherently sequential characteristics. The discovery of the Single Source Shortest Path (SSSP) or the composition of the Minimum Spanning Forest (MSF) of a given graph falls into this category. Kruskal's algorithm is one of the most known algorithms that address the MSF problem. The strictly ordered examination of the graph's edges in order to decide whether they are part of the MSF or not, prohibits the usage of well known parallel strategies, like data partitioning. Our approach attempts to overcome the restrictions imposed by the inherently sequential Nature of the algorithm, by using a Ordered Minimum Spanning Tree (OMST). Let G = (V, E) be a graph with |V| = n vertices and |E| = m edges. The graph G can either be directed or undirected. Let c: E -> R^+ be the weight distribution on the edges of G. Let c = c(e) be the weight of the edge e = (v_i, v_j). Given this setting, the task of finding the optimal (least weight) spanning subgraphs which satisfies a given connectivity requirement is a fundamental problem in the area of network design. The input is an integer k, a k-connected graph G = (V, E) and the weight function c(·). The goal is to find a minimum weight If-connected spanning sub graph of G. In general, the graph connectivity problems come in two flavors: the k-edge connectivity problems (k-Ecss) and the k-vertex connectivity problems (k-VCSS). In undirected graphs the k-edge connectivity problem is to find a minimum cost spanning subgraph in which at least k edge disjoint paths are there between every pair of vertices. In k-vertex connectivity problem we are required to find a minimum cost spanning sub graph in which at least k vertex disjoint paths are there between every pair of non-adjacent vertices. For k = 1, the problem reduces to the problem of finding a minimum spanning tree for the graph, which can be solved exactly in polynomial time, using, for example Kruskal's or Prim's algorithms. For k ≥2, this problem is known to be NP-complete [GJ79], even when the weights are all identical (i.e., the unweighted case). A graph G = (V,E) is called k-edge connected (k-EC) if it contains at least k edge disjoint paths - i. e. paths that do not share an edge between every two vertices. This is equivalent to say that every set of vertices, except Ø and V, is entered by at least k edges. A natural problem is how to find a minimum cost k-edge connected spanning sub graph (k-ECSS) of G, that is, a k-EC spanning sub graph containing the smallest possible number of edges, or having the smallest possible total weight, if the graph is weighted. One can think about this problem as the task of designing a fault-tolerant communication network between several hosts, which means the network should survive a certain number of link failures, i. e. every host should still be able to communicate

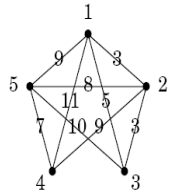
ISSN: 2277-3754
ISO 9001:2008 Certified
International Journal of Engineering and Innovative Technology (IJEIT)
Volume 3, Issue 2, August 2013

with any other. The problem comes in several flavors: the graph can be directed or undirected, the edges can be weighted or not. Further, we can allow parallel edges (multi graphs) or require all edges to be simple (simple graphs).

II. COMPUTATION OF MINIMUM SPANNING TREE

The fig (1) shows matrix on the left below corresponds to the weighted graph on the right. Using Kruskal's algorithm, we iteratively select the cheapest edge not creating a cycle. Starting with the two edges of weight 3, the edge of weight 5 is forbidden, but the edge of weight 7 is available. The edge of weight 8 completes the minimum spanning tree, total weight 21. Note that if the edge of weight 8 had weight 10, then either of the edges of weight 9 could be chosen to complete the tree; in this case there would be two spanning trees with the minimum value.

$$\begin{pmatrix} 0 & 3 & 5 & 11 & 9 \\ 3 & 0 & 3 & 9 & 8 \\ 5 & 3 & 0 & \infty & 10 \\ 11 & 9 & \infty & 0 & 7 \\ 9 & 8 & 10 & 7 & 0 \end{pmatrix}$$



(a) (b)

Fig 1(A) Matrix Representation of Weighted Graph of Fig 1(B)

Theorem-1: A weighted graph with distinct edge weights has a unique minimum weight spanning tree (MST).

Proof 1 (properties of spanning trees). If G has two minimum weight spanning trees, then let e be the lightest edge of the symmetric difference. Since the edge weights are distinct, this weight appears in only one of the two trees. Let T be this tree, and let T` be the other. Since e € E (T) - E (T`), there exists e`€ E (T`) - E (T) such that T`+ e- e` is a spanning tree. By the choice of e, w (e`) > w (e), Now w (T` +e – e`) < w (T`), contradicting the assumption that T` is an MST. Hence there cannot be two MSTs. **Proofs 2** (Kruskal's Algorithm): In Kruskal's Algorithm, there is no choice if there are no ties between edge weights. Thus the algorithm can produce only one tree. We also need to show that Kruskal's Algorithm can produce every MST. The proof in the text can be modified to show this; if e is the first edge of the algorithm's tree that is not in an MST T`, then we obtain an edge e` with the same weight as e such that e`€ E (T`) - E (T) and e` is available when e is chosen. The algorithm can choose e` instead. Continuing to modify the choices in this way turns T into T`.

Theorem-2: No matter how ties are broken in choosing the next edge for Kruskal's Algorithm, the list of weights of a minimum spanning tree (in non decreasing order) is unique. We consider edges in non decreasing order of cost. We prove that after considering all edges of a particular cost, the vertex sets of the components of the forest built so far is the same independent of the order of consideration of the edges of that cost. We prove this by induction on the number of different cost values that have been considered. At the start, none have been considered and the forest consists of isolated vertices. Before considering the edges of cost x, the induction hypothesis tells us that the vertex sets of the components of the forest are fixed. Let H be a graph with a vertex for each such component, and put two vertices adjacent in H if G has an edge of cost x joining the corresponding two components. Suppose that H has k vertices and l components. Independent of the order in which the algorithm consider the edges of cost x, it must select some k - l edges of cost x in G, and it cannot select more, since this would create a cycle among the chosen edges.

Theorem-3: Among the cheapest spanning trees containing a spanning forest F is one containing the cheapest edge joining components of F. Let T be a cheapest spanning tree containing F. If e ≠ E (T), then T + e contains exactly one cycle, since T has exactly one u,v-path. Since u, v belongs to distinct components of F, the u,v-path in T contains another edge e` between distinct components of F. If e` costs more than e, then T` = T – e` + e is a cheaper spanning tree containing F, which contradicts the choice of T. Hence e` costs the same as e, and T` contains e and is a cheapest spanning tree containing F. Applying this statement at every step of Kruskal's algorithm proves that Kruskal's algorithm finds a minimum weight spanning tree.

Theorem-4: If T is a minimum spanning tree of a connected weighted graph G, then T omits some heaviest edge from every cycle of G.

Proof 1 (edge exchange). Suppose e is a heaviest edge on cycle C. If e € E (T), then T - e is disconnected, but C - e must contain an edge e` joining the two components of T - e. Since T has minimum weight, T – e + e` has weight as large as T, so w (e`) ≥ w (e). Since e has maximum weight on C, equality holds, and T does not contain all the heaviest edges from C.

Proof 2 (Kruskal's algorithm). List the edges in order of increasing weight, breaking ties by putting the edges of a given weight that belong to T before those that don't belong to T . The greedy algorithm (Kruskal's algorithm) applied to this ordering L yields a minimum spanning tree, and it is precisely T. Now let C is an arbitrary cycle in G, and let e_1, e_2, …., e_k be the edges of C in order of appearance in L; e_k = uv is a heaviest edge of C. It suffices to show that e_k does not appear in T. For each earlier edge e_i of C, either e_i appears in T or e_i is rejected by the greedy algorithm because it completes a cycle. In either case, T contains a path between the endpoints of e_i. Hence when the algorithm considers e_k , it has already selected edges that form paths joining the endpoints of each other edge of C. Together, these paths form a u,v-walk, which contains a u,v-path. Hence adding e_k would complete a cycle, and the algorithm rejects e_k.

III. THE BASICS OF KRUSKAL'S ALGORITHM

Kruskal's algorithm is one of the most known algorithms for discovering the MSF of an undirected graph with real-valued weighted edges. Specifically, let G=(V; E) be an

ISSN: 2277-3754
ISO 9001:2008 Certified
International Journal of Engineering and Innovative Technology (IJEIT)
Volume 3, Issue 2, August 2013

undirected graph with n=|V| vertices and m=|E| edges, and w:E→R a weight function assigning real-valued weights to the edges of G. The MSF of a given graph is an acyclic subset T of the edges that connects all the vertices that have at least one path between them and at the same time is of minimum weight. The algorithm examines each edge at an ascending order (beginning with the one with the minimum weight) and checks whether it would create a cycle if it was added to the MSF. If this is the case then the edge is discarded, otherwise it is included into the MSF. For our study we select the asymptotically fastest implementation of Ordered Minimum Spanning Tree (OMST), which uses disjoint-set data structures and employs union-by-rank and path compression heuristics. Each disjoint-set is used to store the vertices that belong to a single tree of the OMSF at any given time of the execution. A formal representation of the algorithm is given in Algorithm 1. In this Algorithm we initialize T=0(Line no 1), then we will start construction of Tree, for construction of tree we will start 3 for loops (Line no 2,3 and 4). In Line 5 checks the base condition. In first interaction k=1, i=1,j=1 then it will check a[i][j] is there any weight have 1, if yes add to the spanning tree and also its check's is it p reach's to n-1 edges(Line 5) if yes come out of all loops then returns to Minimum Spanning Tree else increase k then check any weight equal to k if yes add to Spanning Tree, this one continue up to n-1 edges.

IV. ALGORITHM 1: ORDERED MINIMUM SPANNING TREE (OMST) ALGORITHM

Input: Undirected graph G=(V; E), weight function w:E→R

Output: Minimum Spanning Tree

/* Initialization phase */

1. T=∅;
2. p=1; //Initial value of P to check Upper Bond of OMSP
3. foreach k= 1 to n
4. foreach i = 1 to n
5. foreach j = 1 to n
6. if ((k==a[i][j]) && (p!=n-1))
7. {
8. T=T∪ {(i,j)}
9. UNION(i,j);
10. p=p+1;
11. }
12. Return T;

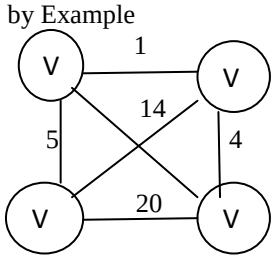
ANALYSIS

Initialize the set A: O(1)
First for loop: |V| to check weights
Second for loop: |V| MAKE-SETs
Third for loop: O(E) FIND-SETs and UNIONs

Assuming the implementation of disjoint-set data structure, that uses union by rank and path compression: O((V + E) α(V)) + O(E lg E)
Since G is connected, |E| ≥ |V| − 1⇒ O(E α(V)) + O(E lg E).

- α(|V|) = O(log V) = O(log E).
- Therefore, total time is O($|V|^3$).

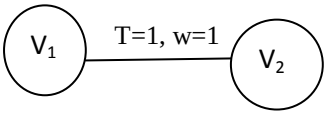
Ex: Outline by Example



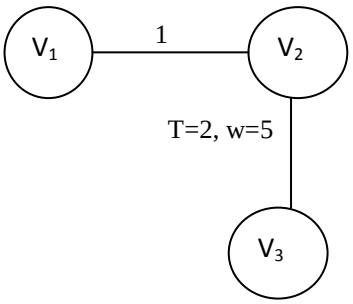
Adjacency Matrix:

$$\begin{bmatrix} 0 & 1 & 14 & 5 \\ 1 & 0 & 4 & 20 \\ 14 & 4 & 0 & 99 \\ 5 & 20 & 99 & 0 \end{bmatrix}$$

Step 1: In the graph, the Edge (i, j) is 1. Either vertex i or vertex j could be representative. Lets choose vertex i.



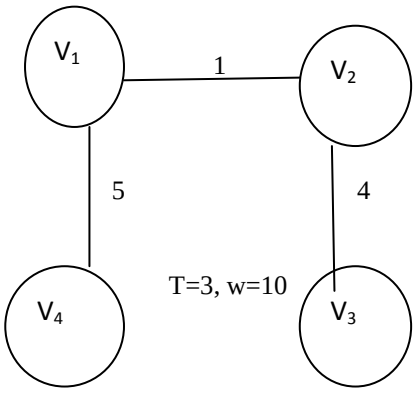
Step 2: Then check any vertex have a weight 1 if yes add edge to Spanning Tree else, increase k by 1 mean now k=2, if ((a[i][j]==2)&& p!=n-1) then add to ST, else increment k by 1 and this process continue up to p reaches to n-1.



4

When k=4 our base case is true if (a[i][j]==4 and p!=n-1(n=4)) so T=2 and w=5.

Step 3: When k=5 our base case is true if(a[i][j]==5 and p!=n-1(n=4)) so T=3 and w=10



ISSN: 2277-3754
ISO 9001:2008 Certified
International Journal of Engineering and Innovative Technology (IJEIT)
Volume 3, Issue 2, August 2013

Step 4: When k=14 our base case is false if (a[i][j]==14 and p!=n-1(n=4)) in a[i][j]==14 is true but p!=n-1(n=4) is false so exit for loop return Ordered Minimum Spanning Tree (OMST).

V. CONCLUSION

We are representing new approach to find minimum Tree in ordered way. This is very simple to understand and to implementation. This approach executes sequentially in a faster way, using data structure so we can access data very efficiently. The drawback of this algorithm takes one extra loop it's so its execution is time consuming. We will improve this in future by reducing that extra loop.

REFERENCES

[1] S. Khuller, Approximation algorithms for finding highly connected sub graphs, In Approximation algorithms for NP-hard problems, Ed. D. S. Hochbaum, PWS publishing co., Boston, 1996.

[2] S. Khuller and B. Raghavachari, Improved approximation algorithms for uniform connectivity problems, J. of Algorithms 21, 1996, 434–450.

[3] E. Lawler. Matroid intersection algorithms. In Math. Programming 9, pages 31{56, 1975.

[4] V. Ramachandran, Fast parallel algorithms for reducible ow graphs, Concurrent Computations: Algorithms, Architecture and Technology, S.K. Tewksbury, B.W. Dickinson and S.C. Schwartz, ed., Plenum press, New York, NY, 1988, pp.117-138; see also Fast and processor efficient parallel algorithms for reducible flow graphs, Tech. Report ACT-103, November 1988, Coordinated Science Laboratory, University of Illinois, Urbana, Illinois, IL 61801.

[5] Arora, S. [1998]: Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems. Journal of the ACM 45 (1998), 753–782

[6] P. Krysta, V.S.A. Kumar, Approximation algorithms for minimum size 2-connectivity problems, in: Proceedings of the 18th International Symposium on Theoretical Aspects of Computer Science, Dresden, Germany, 15–17 February 2001.

[7] R. Jothi, B. Raghavachari, and S. Varadarajan. A 5/4-approximation algorithm for minimum 2-edge-connectivity. In SODA, pages 725–734, 2003.

[8] S. Khuller and R. Thurimella. Approximation algorithms for graph augmentation. J. of Algorithms, 14:214–225, 1993.

[9] H. Nagamochi. An approximation for finding a smallest 2-edge connected subgraph containing a specified spanning tree. Discrete Applied Mathematics, 126:83–113, 2003.

[10] Harold N. Gabow. Approximating the smallest k-Edge Connected Spanning Sub graph by LP-Rounding, 2009.

[11] Jaewon Oh, Iksoo Pyo and Massord Pedram. Constructing Minimal Spaaning/ Steiner Trees with Bounded Path Length, pp 244-249, 1996.

[12] David Pritchard. K-Edge-Connectivity: Approximation and LP Relaxation, 2010.

[13] Xiaofeng Han, Pierre Kelsen, Vijaya Ramachandran and Robert Tarjan. Computing Minimal Spanning Subgraphs in Linear Time, 1995.

[14] Artur Czumaj and Andrzej Lingas. On Approximating of the Minimum-Cost k-Connected Spanning Sub graph Problem.

[15] Dominik Alban Scheder. Approaches to Approximating the minimum weight k-Edge connected spanning sub graph of a mixed graph, 2006.

[16] J.B.Kruskal. On the shortest spanning sub tree of a graph and the travelling salesman problem. Proceedings of the American Mathematical Society, Volume 7, pp. 48-50, 1956.