Decomposable Algorithms for Computing Minimum Spanning Tree

Ahmed Khedr and Raj Bhatnagar

Univ. of Cincinnati, Cincinnati OH 45221, U.S.A.,
akhedr@ececs.uc.edu, Raj.Bhatnagar@uc.edu

Abstract. In the emerging networked environments computational tasks are encountering situations in which the datasets relevant for a computation exist in a number of geographically distributed databases, connected by wide-area communication networks. A common constraint in such situations of distributed data is that the databases cannot be moved to other network sites due to security, size, privacy or data-ownership considerations. For these situations we need algorithms that can decompose themselves at run-time to suit the distribution of data. In this paper we present two such self-decomposing algorithms for computing minimum spanning tree for a graph whose components are stored across a number of geographically distributed databases. The algorithms presented here range from low granularity to high granularity decompositions of the algorithms.

1 Introduction

Minimum Spanning Tree: The minimum weight spanning tree (MST) problem is one of the most typical and well-known problems of combinatorial optimization. Minimum spanning tree algorithms for single and parallel processor architectures have been studied and analyzed extensively [3, 5, 6, 8]. In the last two decades efforts have been concentrated on developing faster algorithms based either on more efficient data structures or multiple processors. Karger et al. in [7] propose a linear expected-time algorithm. In many MST algorithms designed for closely coupled processor systems, a number of processors are assigned to work on a shared dataset and they work together under the specification of a parallel algorithm [1, 2, 9]. Gallager *et al* [4] present a distributed algorithm that constructs the minimum weight spanning tree in a connected undirected graph with distinct edge weights. A processor is assigned to each vertex of the graph, knowing initially only the weights of edges incident on this vertex. The total number of messages to be exchanged among the processors for a graph of N vertices and E edges is at most $5N\ logN + 2E$ where each message contains at most one edge weight and $\log 8N$ bits.

Widely Distributed Knowledge: In environments of geographically distributed data and knowledge, subgraphs may reside on different sites of a communication network. Communication time across the sites of a wide-area network is orders of magnitude larger than that for the processors of a set of closely-coupled

processors. The closely coupled processors' model is, therefore, not applicable in this situation. The desired model is one in which the processor at each site performs large amount of computation with its own subgraph and periodically exchanges minimal amount of necessary information with processors at other sites to construct a correct global solution.

Algorithms for processing such distributed graph structures have received very little attention. The emerging networked knowledge infrastructure requires algorithms for such distributed data situations. The focus of research described in this paper is on such situations of widely distributed knowledge.

The situation addressed by us is very different from the one addressed by algorithms using Distributed Shared Memory (DSM). Two major differences from the DSM environments are: (i) In DSM model the frequent messages from one processor to the other are for reading and retrieving fine-grained data whereas in our model results of fairly large local computations are exchanged very infrequently among the processors. This is because the communication cost is overwhelming in the situations being modeled by us. (ii) The algorithms using DSM seek to minimize the number of participating processors whereas in our algorithms the number of participating databases, and hence the number of processors, are determined by the global problem to be solved and we seek to minimize the number of messages that need to be exchanged among them.

An Example Scenario We briefly describe here a real-life situation in which the decomposable algorithm would be very useful. Consider a number, say n, of airlines each of which has its own database about flight segments it operates. All the flight segments operated by an airline may be represented as a graph, and one such graph exists in the computer system operated by each airline. Some cities are served by more than one airline and some may be served by only one of them. Therefore, any two subgraphs may have some shared vertices.

Consider the situation in which these airlines decide to collaborate. Such collaborations do not mean a complete merger of their routes and operations but only sharing each other's flight segments to complete flight paths for customers. The *costs* along each edge of a subgraph may reflect price, seat availability, travel time etc. Therefore, it is not possible for an airline to share a fixed subgraph (along with edge costs) with the other airlines. It is required that the databases consult each other to determine the least cost paths whenever such paths are needed. The dynamic nature of costs along the edges of subgraphs requires frequent computation of least cost paths across the global graph formed by all the component subgraphs. An algorithm that can dynamically consult all the individual databases to infer the minimum cost paths is desirable, and our algorithms seek solutions for these and similar problems.

2 Algorithm Decomposition

The abstraction for the problem addressed by us in this paper can be described as follows. We assume that each of a number of networked data sites contains a subgraph. For each site, an implicit global graph exists and it is formed by

combining the local subgraphs at all those data sites with which it can communicate. We present algorithms for computing minimum spanning tree in these implicit global graphs, without having to explicitly construct the global graph, and also with minimum communication among the data sites so as to preserve communication resources and data security at individual sites.

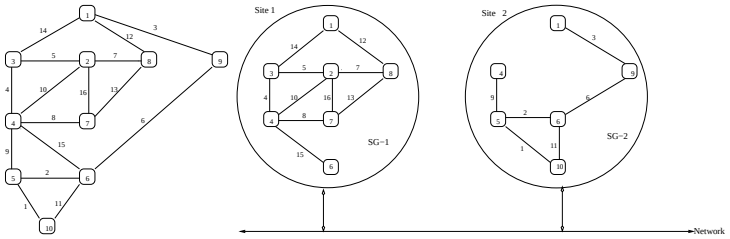
Our problem involves design of algorithms for computing minimum spanning tree in a graph that is stored as overlapping component subgraphs across various sites of a network. We operate under the constraint that data cannot be transferred between sites. The mathematical formulation of our problem can be described as follows:
Let us say a result R is obtained by applying a function F to a dataset $\mathcal{D}$ that is: $R = F(\mathcal{D})$

In the case of algorithms for distributed databases, $\mathcal{D}$ (the global graph) cannot be made explicit and is known only implicitly in terms of the explicit components $D_1, D_2, \ldots, D_n$. The implementation of F in the last equation can be redesigned by an equivalent formulation: $R = G(g_1(D_1), g_2(D_2), \ldots, g_n(D_n))$ That is, a local computation $g_i(D_i)$ is performed at $Site_i$ using the database D_i. The results of these local computations are aggregated using the operation G.

Granularity of Function Decomposition It may not be mathematically possible to decompose every function F into an equivalent set of G and g_i functions. In those cases, we consider a sequential algorithm for evaluating F which is a step-sequence composed of simpler functions as in the sequential steps of an algorithm to evaluate F. Thus, we represent F as: $F = f_1; f_2; \ldots; f_n$. Then, the functional decomposition can be found for each f_i step and the entire sequence of f_is is evaluated in a decomposed manner, one f_i at a time, to evaluate F. When the function F is directly decomposable into a set of G and g_i operators, we refer to it as a higher granularity decomposition. The more primitive the level of f_is employed to achieve the decomposition, the lower is the granularity of decomposition. Evaluation of each f_i may require exchange of some messages among the sites. Therefore, it is highly desirable to perform the decomposition of F at as high a level of granularity as possible. The algorithms we present in this paper range from low granularity to higher granularity. It is expected that the higher granularity decompositions would be performed with lower communication costs.

Distributed Representation The representation of a graph for the cases of closely coupled processors and the widely distributed sites can be compared as follows. In the first case a graph $\mathcal{G}$ with its vertices V and edges E is stored either on a single computer in the form of an adjacency table or in the shared memory addressable by each of the closely coupled processors. In a widely distributed environment overlapping components of a graph may be stored on different sites of a network. Figure 1 below shows an example graph that may be stored on a single site.

Second part of Figure 1 shows two graphs that may be stored on two different sites of a network. The two subgraphs together constitute the same graph as in



(a) Complete Graph

(b) Two Subgraphs

Fig. 1. An implicit complete graph is formed by multiple component graphs stored at different network sites.

Figure 1 and share some of the vertices. This situation can be easily generalized to n network sites, each containing a part of the complete graph.

One major assumption made by the *closely coupled processors* paradigm is that the data can be efficiently moved around among the processors and can be made available very quickly to a processor site that needs it. This assumption does not hold good in the knowledge environment where potentially huge databases exist at geographically dispersed sites, connected by wide-area networks. From the perspective of elapsed time, it is very expensive to move data around such networks. The alternative for us is to design algorithms that can decompose themselves according to the distribution of data across the sites. Partial computations resulting from such decompositions can then be sent to the sites where data resides and the results obtained then combined to complete the desired computations. Since a different decomposition of an algorithm may be required for each type of distribution of data across sites, we would like to develop algorithms that can decompose themselves into relevant partial computations to suit the distribution of data. The algorithms should also minimize the communication cost among the sites. For example, we may want to determine the edges that should be retained in a minimum cost spanning tree of the complete graph by the network sites having to exchange minimum number of messages among themselves. The vertices and edges of the two subgraphs of Figure 2 above can be represented by the following two tables at their respective sites:

Complexity Cost Models We choose the following three cost models for analyzing the complexity of our algorithms.

$vertex_1$	$vertex_2$	$weight$
1	8	12
4	3	4
4	6	15
1	3	14
3	2	5
2	8	7
2	7	16
7	4	8
4	2	10
8	7	13

Table 1. Database at Local Site 1

$vertex_1$	$vertex_2$	$weight$
9	1	3
9	6	6
6	5	2
5	4	9
5	10	1
10	6	11

Table 2. Database at Local Site 2

Cost Model #1 Communication Cost only: In this cost model we count the number of messages, N_m, that must be exchanged among all the participating sites in order to complete the execution of the algorithm. One message exchange includes one message sent by a site requesting some computation from another site and the reply message sent by the responding site. This cost model is relevant in situations where: (i) We need to analyze the number of messages exchanged because some critical resource, such as the battery power of sensors, is exhausted by the sending of messages; and (ii) Communication cost of messages is orders of magnitude larger than the cost of computations at local sites.

Cost Model #2 Communication + Computation Cost: In this model we examine a weighted sum of the number of messages exchanged and the number of local operations performed. If N_m messages are exchanged among the sites and a total of N_c computational units are performed at all the sites combined then the algorithm's cost is given by: $a * N_m + b * N_c$ where a and b are the weights representing the relative costs of exchanging a message and performing a local computational unit. This cost model is useful when the local computation time within a site is not negligible and must be included within the cost model.

When the databases stored at the sites are huge, as in many scientific and data-mining applications, the time to execute a local computation may be comparable to the time taken for exchanging a message across a wide-area network.

Cost Model #3 Elapsed Time Cost: In this model we examine a weighted sum of the number of messages exchanged and the number of local operations performed, while accounting for parallel transmission of messages and simultaneous execution of local computations at the participating sites. If N_m messages are exchanged among the sites and a total of N_c computational units are performed at all the sites combined then the algorithm's cost is given by: $(a*N_m+b*N_c)/p$ where a and b are the weights representing the relative costs of exchanging a message and performing a local operation, and p is the average number of messages that can be exchanged in parallel. This cost model is useful when our criterion is the total elapsed time for executing the algorithm.

3 Low Granularity Decomposition

Here we start by adapting Prim's algorithm and decomposing each of its steps. Prim's algorithm works by building a tree which starts from an arbitrary root vertex r and grows until the tree spans all the vertices in V (V is the graph vertex set). At each step, an edge connecting a vertex in V_1 to a vertex in $V - V_1$ is add to the tree. This strategy is *greedy* since the tree is augmented at each step with an edge that contributes the minimum amount possible to the tree's weight. For a decomposable version of Prim's algorithm we implement each small step of the above algorithm in its decomposable form. Therefore, this is the lowest granularity decomposition. In this version we do not accumulate any work for a single node to perform and exchange results only after a bigger local result or summary has been generated and needs to be exchanged with neighboring subgraphs.

3.1 Algorithm Outline

Input: a connected graph with a vertex set V ($|V| = m$) and edge set E divided into n parts, $SG_1(V_1, E_1)$,$SG_2(V_2, E_2)$ ... $SG_n(V_n, E_n)$, each part residing at a different site and any two subgraphs may share some vertices, r The tree root (a vertex of V), W weight functions on $E_1, E_2, \ldots E_n$, $InTheTree[1:m]$ global array initialized to F, and $Nearst[1:m]$ global array initialized to ∞ except for, $Nearst[r]$ which is set to 0.

Output: $Parent[1:m]$ global array of a minimum spanning tree.

$MessageCounter = 0$
$Parent[r] = 0$
for step = 1 to $m - 1$ do
  select a vertex u that minimizes $Nearest[u]$ over
  all u such that $InTheTree[u] = F$
  set $InTheTree[u] = T$
  send message to every $site_i$

select all vertices ($vertex_2$), $weight$ from SG_i where
$vertex_1$=u, and there is an edge
between $vertex_1$ and $vertex_2$
$MessageCounter = MessageCounter + n$
Results = $vertex_2$, $weight$ from SG_i
while(R = NextElement(Result))
where R is an object from local sites contains $vertex_2$
and weight, and NextElement is a function take
element by element from the local site results
v = R.$vertex_2$ and w = R.weight
if InTheTree[v] = F then
Update $Nearest[v]$ and $Parent[v]$ for all
$v \in V$ that are adjacent to u
if ($Nearest[v]$) > w)
$Nearest[v] = w$
$Parent[v] = u$
end if
end if
end while
end for

3.2 Complexity Analysis

According to the cost models we defined in section 2 we derive below an expression for the number of messages that need to be exchanged for our algorithm dealing with the implicit set of tuples. Let us say: (i) There are a total number of m vertices in the whole graph and (ii) There are n relations, $D_1 \ldots D_n$, residing at n different network sites.

Cost Model #1: In this cost model we count the number of messages, N_m, that must be exchanged among all the participating sites in order to complete the execution of the algorithm. In this case the complexity can be explained as the following. We maintain a boolean array *InTheTree* to keep track of the vertices in/not in the tree. We then select the next edge uv to be added to the tree by maintaining an array $Nearest[1 : m]$ at the algorithm initiating site where $Nearest[v]$ is the distance from the vertex u to v in the tree and $Nearest[v] = \infty$ if v is not yet in the tree. In the algorithm we have $m - 1$ stages, and for each stage the querying site needs to send only one message to every other site. Therefore, a total of $n * (m - 1)$ messages will have to be exchanged to complete this algorithm.

Cost Model # 2: In this model we examine a weighted sum of the number of messages exchanged and the number of local operations performed. For each exchanged message, this algorithm performs one SQL query at the responding site. Therefore, the total cost for the algorithm will be $a * (m-1) * n + b * (m-1) * n = (a+b)n(m-1)$ where a is the time taken to exchange a message and b is the average time taken to perform a simple SQL query at a site.

Cost Model # 3: In this model we examine a weighted sum of the number of messages exchanged and the number of local operations performed, while discounting the effects of messages and operations that can be executed in parallel, simultaneously at different sites. Therefore, the total cost for the algorithm will be $(a*(m-1)*n\ +\ b*(m-1)*n)/n = (a+b)(m-1)$ where a is the time taken to exchange a message and b is the average time taken to perform an SQL query at a site.

4 Higher Granularity Algorithm for MST

The previous section take an algorithm for a single graph and decompose its steps to achieve a decomposable version. In this section we present an algorithm designed specially for the distributed environments. This algorithm works by taking each shared vertex at each local site as a fragment. We grow these fragments by adding the minimum outgoing edges this is described in **Local Computation** section below. Then we combine these fragments by the global computations described in **Global Computation** section below.

Fragment Definitions and Properties Before we present the second algorithm we introduce some definitions like fragment and outgoing edge.

Definition 1 A fragment f of an MST is a subtree of the MST.
Definition 2 An outgoing edge e of a fragment f is an edge that has one of its vertices in f and the other vertex out of f.
Property 1 Given a fragment of an MST, let e be the minimum weight outgoing edge of the fragment. Then joining e and its adjacent non-fragment vertex to the fragment yields another fragment of an MST. This result in the context of fragments has been shown in [4] and we briefly outline below their proof for this property.

Suppose the added edge e is not in the MST containing the original fragment. Then there is a cycle formed by e and some subset of the MST edges. At least one edge $x \neq e$ of this cycle is also an outgoing edge of the fragment, so that $W(x) \geq W(e)$ where $W(x)$ is the weight associated with the edge x. Thus, deleting x from the MST and adding e forms a new spanning tree which must be minimal if the original tree was minimal. Therefore, the original fragment including the outgoing edge e is a fragment of the MST.

We now present a self-decomposing version of an MST building algorithm for a graph that exists in parts across a network. The assumptions describing the situation are as follows.

- The complete global connected graph $\mathcal{G}$ is represented by n distinct but possibly overlapping subgraphs. Each subgraph is embedded in a database D_i residing at a different computer system (site) in the network. Each subgraph consists of vertices and edges such as: $SG_1(V_1, E_1), SG_2(V_2, E_2) \ldots SG_n(V_n, E_n)$ and $\mathcal{G} = \bigcup_i\ SG_i$.
- Each subgraph SG_i may share some vertices with other subgraphs. The set of vertices shared by SG_i and SG_j are: $V_{sh}(i,j) = V_i \bigcap V_j$.

- The set of all vertices that are shared by at least two subgraphs is: *Shared* $= \bigcup_{i,j} \ V_{sh}(i,j)$.
- Each edge e of a subgraph SG_i has a weight $W(e)$ associated with it.
- The weight of a tree in the graph is defined as the sum of the weights of the edges in the tree and our objective is to find the global tree of minimal weight, that is, the global MST.

4.1 Algorithm Outline

Data Structure: A table called *Links* is maintained at the coordinator site and it stores information about candidate edges for linking various fragments to each other in order to complete the MST. This table has one row and one column for each shared vertex in the global graph. Each shared vertex name in the table is a representative of all the fragments, on all the different sites, that contain this shared vertex. Each entry in the table represents the site number and the weight of a potential edge that can link the row-fragment to the column-fragment. How these values are updated is given in the *Global Computations* paragraph below.

Local Computations The following steps are executed at each site on its locally resident subgraph. The goal is to create fragments within each subgraph at its local site.

- If the site has r shared vertices then each shared vertex is initialized as a separate fragment. That is, $f_i = v_i, i = 1, 2, \ldots, r$ and v_i is the i^th shared vertex.
- Expand fragments f_i's as follows
  - For each fragment f_i find its minimum weight outgoing edge e_i such that e_i does not lead to a vertex already included in some other fragment.
  - From among all the outgoing edges select the one with least weight. That is, $e_{min} = argmin_i \, W(e_i)$.
  - Add e_{min} to the fragment that selected this edge as its minimum outgoing edge.

Global Computations Each site, after completing its local computations, sends a message to the coordinator site informing it that the site is ready for global level coordination. After all the participating sites are ready the coordinator site performs the following steps to generate the global MST. In the final state each site knows the edges in its local subgraph that are included in the global MST. A complete MST is not generated at any single site.

- At each site, find the minimum-weight outgoing edge from each fragment to every other locally resident fragment. Each fragment is identified by the unique shared vertex contained in it. This step generates a tuple $<$*site-number, from-fragment, to-fragment, weight* $>$ for each fragment at the site. All these tuples are sent to the coordinator site.
- Update the cell *Links(row,column)* table with the above tuples received from all the sites as follows.

- Include all the received tuples from the local sites in a set S where
  row = *from-fragment*; and
  column = to-fragment
- From the set S select the tuple t with the minimum weight value.
- Assign this tuple to *Links(row,column)*.

– Repeat the following steps till the *Links* table contains only one row and one column:
  - Select in *Links* the cell with minimum value. This edge links its row-fragment (f_{row}) to its column-fragment (f_{col}) and is now selected to be in the global MST. This step also means that the fragments f_{row} and f_{col} are now merged into a new fragment, called $f_{row-col}$.
  - Update the *Links* tuple as follows:
    * Create a new row and a new column in *Links* for the fragment $f_{row-col}$ and delete the rows and columns for the original two fragments f_{row} and f_{col}. The value for the cells in the new row and column can be determined as:
      · For every shared vertex d in the table if the minimum weight of outgoing edge from d to f_{row} is v_1 and to f_{col} is v_2 and $v_1 < v_2$ then the minimum weight of outgoing edge from d $f_{row-col}$ will be v_1.

End Algorithm.

4.2 Algorithm Correctness

An algorithm that grows and combines fragments until an MST for the graph is formed is given by Gallager et al. in [4]. Their algorithm is designed for a parallel processing environment in which one processor can be assigned to work at each vertex of the graph. Since our situation is different, we show below that our algorithm for growing and combining fragments will yield the global MST.

The set $Shared-Vertices$ contains all those vertices of the subgraphs that exist in more than one subgraph. We grow fragments starting from each member of $Shared–Vertices$ that exists at any site. That is, if k shared vertices occur in a subgraph at a site then the site will locally generate three distinct fragments each containing one and only one shared vertex. Then we combine these fragments by the global computations described in section 4.1 above.

We now show that (i) locally generated fragments are parts of the minimum spanning tree and (ii) the edges selected to combine local fragments are also part of the global MST.

Assertion: Each locally generated fragment is a part of the global MST.

Proof: The algorithm ensures that each fragment contains one and only one vertex that is shared with subgraphs at other sites. All the other vertices included in a fragment are completely local to the subgraph and the site on which the fragment resides. Therefore, in the global graph there does not exist an edge that connects a vertex in a local fragment to a vertex in another fragment that resides at some other site. This implies that edges between vertices of a fragment

cannot be replaced by edges existing in some other subgraph residing at some other site.

Therefore, as long as a fragment is an MST of the local subgraph, it will be a fragment of the global MST.

Now we show that a fragment is part of the MST of its local subgraph i.e. each added minimum outgoing edge e is part of the MST.

Suppose the added edge e is not in the MST containing the original fragment. Then there is a cycle formed by e and some subset of the MST edges. At least one edge $x \neq e$ of this cycle is also an outgoing edge of the fragment, so that $W(x) \geq W(e)$ where $W(x)$ is the weight associated with the edge x. Thus, deleting x from the MST and adding e forms a new spanning tree which must be minimal if the original tree was minimal. Therefore, the original fragment including the outgoing edge e is a fragment of the MST.

Assertion: Edges selected to combine local fragments are part of the global MST.

Proof: Each fragment contains one and only one shared vertex. Therefore, joining two fragments by an edge is equivalent to bringing two shared vertices into one connected component of the MST. An edge that connects two shared vertices (and their respective fragments) may exist on more than one site. Our algorithm examines all possible sites on which the same two shared vertices (and their fragments) may be connected and selects the minimum-weight edge from among all the candidate edges on all the sites.

Now, if this does not result in the MST then there must exist an edge on a site that links two fragments such that it results in a lower weight MST. Suppose e_{ij} which is least weight minimum outgoing edge from fragment f_i to fragment f_j chosen from *Links* table is not the correct minimum outgoing edge from from fragment f_i to fragment f_j. Then there is another path from f_i to f_j with minimum weight less than the e_{ij} weight. Part of this path is an outgoing edge y from f_i with weight is less than e_{ij} which contradict with the assumption e_{ij} is the minimum outgoing edge from fragment f_i.

4.3 Complexity Analysis

We analyze below the MST finding algorithm for its complexity from the perspective of the three scenarios mentioned in section 2.

Cost Model # 1: In this cost model we count the number of messages, N_m, that must be exchanged among all the participating sites in order to complete the execution of the algorithm. In this case the complexity will be: (i) n exchanged messages to perform local fragments; and (ii) n exchanged messages to combine the local fragments. Then the total number of exchanged messages will be $2a*n$. where a is the time taken to exchange a message

Cost Model # 2: In this model we examine a weighted sum of the number of messages exchanged and the number of local operations performed. For each exchanged message, this algorithm performs one SQL query at the responding site. Therefore, the total cost for the algorithm will be $2(a+b)*n$ where a

and b are the weights representing the relative costs of exchanging a message and performing a local operation.

Cost Model # 3: In this model we examine a weighted sum of the number of messages exchanged and the number of local operations performed, while discounting the effects of messages and operations that can be executed in parallel, simultaneously at different sites. Therefore, the total cost for the algorithm will be $2(a+b) * n/n = 2(a+b)$ where a and b are the weights representing the relative costs of exchanging a message and performing a local operation.

5 Conclusion

We have demonstrated that it is possible to execute graph operations for databases stored across wide-area networks. These algorithms are decomposable at run time depending on the set of shared vertices among the graph components stored at different network sites. We examined the complexity of our algorithms from the perspective of cost models that take into account the communication cost across the sites of a wide-area network. It turns out that these graph algorithms can be computed without too much communication overhead. These algorithms can perform more efficiently than having to bring all the data at one site while still preserving the privacy and ownership value of individual databases.

References

1. Abdel-Wahab, H.; Stoica, I.; Sultan, F.; Wilson, K. *A Simple Algorithm for Computing Minimum Spanning Trees in the Internet* Information Sciences, Volume: 101, Issue: 1-2, September, 1997, pp. 47-69.
2. Kenneth A. Berman and Jerome L. Paul *Fundamentals of Sequential and Parallel Algorithms* PWS Publishing Company 1997.
3. Mcdiarmid, Colin; Johnson, Theodore; Stone, Harold S. *On finding a minimum spanning tree in a network with random weights* Random Structures and Algorithms, Volume: 10, Issue: 1-2, January - March 1997, pp. 187 - 204.
4. Gallager R. G. and et.al A Distributed Algorithm for Minimum- Weight Spanning Trees ACM Transaction on programming and Systems, 5 66-77.
5. Graham, R.L. and P. Hell.*ON the History of the Minimum Spanning Tree Problem* Annals Of the History of Computing,7 1985:43-57.
6. toica, Ion; Sultan, Florin; *Keyes A Hyperbolic Model for Communication in Layered Parallel Processing Environments* journal of Parallel and Distributed Computing, Volume: 39.Issue: 1, November 25,1996, pp.29-45.
7. Karger DR, Klein PN, Targan RE. *A randomized linear -time algorithm to find minimum spanning tree.* Journal of the Association for Computing Machinery ; 42/2:321-8.
8. Nancy A. Lynch *Distributed Algorithms* Morgan Kaufman Publishers,Inc. San Francisco, California 1996.
9. King, Valerie; Poon, Chung Keung; Ramachandran,Vijaya; Sinha, Santanu *An optimal EREW PRAM algorithm for minimum spanning tree verification* Information Processing Letters, Volume: 62, Issue: 3, May 14, 1997, pp. 153-159.