Parallel Minimum Spanning Tree Heuristic for the Steiner Problem in Graphs

Hoda Akbari[1], Zeinab Iranmanesh[1], and Mohammad Ghodsi[1,2*]

[1]*Sharif University of Technology,* [2]*IPM school of Computer Science, Tehran, Iran*
{h_akbari, iranmanesh}@ce.sharif.edu, ghodsi@sharif.edu

Abstract

Given an undirected graph with weights associated with its edges, the Steiner tree problem consists of finding a minimum weight subtree spanning a given subset of (terminal) nodes of the original graph. Minimum Spanning Tree Heuristic (MSTH) is a heuristic for solving the Steiner problem in graphs. In this paper we first review existing algorithms for solving the Steiner problem in graphs. We then introduce a new parallel version of MSTH on three dimensional mesh of trees architecture. We describe our algorithm and analyze its time complexity. The time complexity analysis shows that the algorithm's running time is $O(\lg^2 n)$ *which is comparable with other existing parallel solutions.*

1. Introduction

A great number of the recent applications often require the underlying network to provide multicasting capabilities. Multicast refers to the delivery of packets from a single source to multiple destinations. At the routing level, a multicast routing scheme is responsible for determining the packet delivery path from the source to all destinations, typically a multicast tree [13]. Generation and minimization of the cost of such tree have been traditionally formulated as the *Steiner Tree Problem*. The Steiner Tree Problem involves constructing the least cost tree that spans a given set of points. In addition to multicast routing in communication networks, the Steiner tree problem has numerous applications especially in the areas of telecommunication, distribution and transportation systems. The computation of phylogenetic trees in biology and the routing phase in VLSI design are real life problems that have been modeled as the Steiner tree problem [12]. Another interesting application is in the billing strategies of large telecommunications network service providers. The bill isn't based on the actual number of circuits provided, which may change over time, but on a simple formula calculated for an ideal network which will provide all the facilities at minimum cost. Several other network design problems can be formulated as generalizations of the Steiner tree problem.

Steiner tree problem or so called *Steiner Problem in Graphs* (SPG) is a classic combinatorial optimization problem. Karp showed that its decision version is NP-complete [18], although some well known special cases of the SPG can be solved in polynomial time. When $|N| = 2$ the problem reduces to the shortest path problem while when $\mathbf{N} = \mathbf{V}$ the problem reduces to the minimum spanning tree problem. Both these problems can be solved in polynomial time. On the other hand, the Steiner tree problem is NP-hard when the graph G is a chordal graph, a bipartite graph or a complete graph with edge weights either 1 or 2. Thus in the general case the problem is an NP-hard problem.

Due to its NP-hardness, several heuristics have been developed to approximate its solution. However, there are some major difficulties in deployment and application of the existing algorithms to real-time communication networks. One of these difficulties in distributed solutions is that existing protocols don't support the information needed to be exchanged among nodes. Furthermore the convergence time may be prohibiting for the multicast trees that change frequently. So, it may be a good approach to design a parallel algorithm which can be run very fast on a single node. In this paper we present parallel version of the *minimum spanning tree heuristic* (MSTH) for the Steiner problem in graphs on *three dimensional mesh of trees* (3dMOT) architecture. In the next section, we review the definition of the problem and its variations. In sec. 3 we present a survey of proposed algorithms for the Steiner problem in graphs. In sec. 4 we present our approach for making MSTH parallel on 3dMOT. We describe our algorithm and present its time complexity. Finally, in sec. 5 we make an overall evaluation of the proposed algorithm and conclude the work.

* This author's work was partially supported by a grant from IPM (N. CS2386-2-01.)

2. Problem definition, variations and generalizations

Let $G = (V, E)$ be a connected undirected graph, where V is the set of nodes and E denote the set of edges. Given a non-negative weight function $w: E \to R_+$ associated with graph edges and a subset $X \subseteq V$ of terminal nodes, the Steiner Problem in Graphs, *SPG(V, E, w, X)*, consists of finding a minimum weighted connected subtree of G spanning all terminal nodes in X. The solution of SPG *(V, E, w, X)* is *Steiner minimum tree*. The non-terminal nodes that end up in the Steiner minimum tree are called *Steiner nodes*.

Terminal Steiner Tree Problem is a variation in which all the terminal nodes must appear at leaves of the tree. This problem that is also proved to be NP-complete has been matter of concern because it has direct application in VLSI design [5]. In *Complete Steiner Problem* the input graph is assumed to be complete. Another variation is the *Complete Steiner (1, 2)* in which the input is a complete graph with edge weights 1 or 2. All of these variations are NP-complete [6]. SPG – or its terminal version – are sometimes said to be *metric*, i.e. the triangle inequality holds for edge weights in the input graph. This imposes no limitation on the Steiner problem itself, since we can replace any edge with the shortest path connecting its ends [19, 20]. The *Steiner Network Problem* generalizes the metric Steiner tree problem to higher connectivity requirements: Given a graph *G = (V, E)*, a cost function on edges, and a function *r* mapping unordered pairs of vertices to Z^+, find a minimum cost graph that has *r(u,v)* edge disjoint paths for each pair of vertices *u* and *v* [20]. The issue of multipoint routing for multimedia traffic has led to emergence of *Constrained Steiner Tree Problem*, in which the problem is to find a minimum cost tree such that the delay – delay variation or both – between the source and each of the destinations is bounded. The *Dynamic Steiner Tree Problem* is another generalization of the problem, in which the set of destination nodes changes over time by receiving join or delete requests from nodes, and problem asks for a sequence of optimal trees [11].

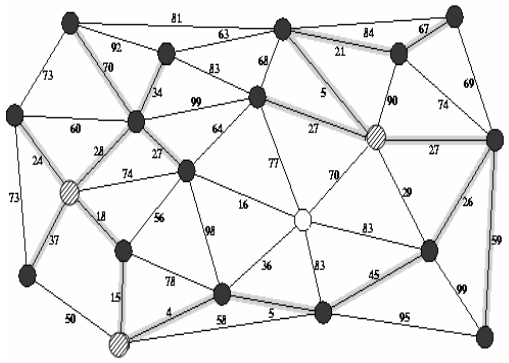

Figure 1. Sample Steiner tree. Black nodes are destination nodes, white nodes and hatched nodes are non-destination. The Steiner nodes are hatched.

3. Related work

As SPG is NP-complete, there is little hope to find a polynomial time solution for it. All the work done to find a solution so far falls into three categories: *Exact Algorithms*, *Approximation Algorithms* and *meta-heuristics*.

Two popular exact algorithms, the *Spanning Tree Enumeration Algorithm* (STEA) which enumerates all possible combinations of Steiner nodes, and the *Dynamic Programming Algorithm* (DPA), present time complexities of $O(p^2 2^{(n-p)} + n^3)$ and $O(3^p n + 2^p n^2 + n^3)$ respectively, where n is the number of nodes in the network and p is the number of multicast members [11]. These algorithms require long computation time or huge computational power for solving bigger problems, like the branch and bound algorithm proposed in [7], that makes use of computational grids.

In [17], the author offers an approximation algorithm with performance ratio 5/3 based on finding a minimum spanning tree in 3-uniform hypergraphs that finds the solution with probability at least 1/2 and claims that the algorithm runs in $O(\lg^2 n)$ time, using $O(n^3)$ processors. The best approximation algorithm known so far is due to Robins and Zelikovsky whose performance ratio is about 1.55 and even better for special cases such as quasi-bipartite and complete(1-2) graphs [19].

Among several heuristics proposed to find an approximate solution, *Traveling Salesman Problem Heuristic* (TSPH), *Minimum Spanning Tree Heuristic* (MSTH), and *Average Distance Heuristic* (ADH) have performance ratio of 2.

TSPH is a heuristic based on the traveling salesman problem (TSP) and involves finding a tour for the graph induced by the network followed by removing the most expensive link.

Shortest path heuristic (SPH) computes the tree by connecting all the terminals to an arbitrary root through their shortest paths and then finding the minimum spanning tree of the graph induced by the union of these paths, repeatedly removing the nonterminal leaves.

The algorithm presented in [11] is a distributed algorithm based on an improved version of the ADH heuristic, known as ADH with Full connection (ADHF). [11] Also provides an efficient approach that supports dynamic multicast membership, by means of periodic improvement of locally inefficient subtrees.

In [9] the author introduces a new algorithm using the *Random Neural Networks* to find potential Steiner vertices that are not already in the solution returned by the MSTH or ADH, starting with the solution of the MSTH or ADH.

The first approximation algorithm for SPG having an approximation ratio constant lower than 2 was due to Zelikovsky [10] with performance ratio 11/6. Then he repeatedly improved this ratio to currently best known performance ratio: 1.55.

The heuristics proposed to find the Steiner tree for routing applications are either centralized or distributed. In the centralized approach, a central node that is aware of the state of the whole network computes the tree. The computation is generally easy and fast. But impractical for large networks where the overhead of maintaining, in a single node, coherent information about the state of the entire network may be prohibitive. In a distributed approach, on the other hand, each node of the network actively contributes to the algorithm computation. Distributed routing algorithms can be slower and more complex than the centralized ones, but they become indispensable when the network nodes can not reach a complete knowledge of the topology and of the state of the network [11].

Some meta-heuristics are proposed as the solution for the Steiner problem in graphs too. Among the most efficient ones, it is found implementations of meta-heuristics such as genetic algorithms, tabu search, GRASP and simulated annealing [16, 18].

Esbensen and Mazumder [8] proposed a genetic algorithm and discuss its application in global routing of VLSI layouts. The algorithm's encoding is based on the use of the Distance Network Heuristic (DNH) which is a deterministic heuristic for the SPG. The performance of algorithm is compared to that of two heuristics from the literature and it has been shown that the algorithm is superior.

Di Fatta, Lo Presti, and Lo Re proposed a parallel genetic algorithm for the Steiner problem in networks. When solving Beasley's OR Library standard test problems, they obtain promising speedup values. Recently, the same authors working with Storniolo and Urso extend their proposal presenting a parallel hybrid method that combines a distributed genetic algorithm and a local search strategy using a specific Steiner tree problem heuristic [1].

Tabu Search was introduced by Glover in 1986. TS is an extension of classical *local search* methods typically used to find approximate solutions to difficult combinatorial optimization problems [3]. Ribeiro and Souza [18] proposed an improved tabu search for the Steiner problem in graphs. The important feature of the algorithm is that move estimations, elimination tests, and neighborhood reduction techniques are used to speedup the local search and lead to a much faster algorithm with similar performance in terms of solution quality. In the context of parallel tabu search for the Steiner problem in graphs, Bastos and Ribeiro [2] describe a two phase algorithm: in their approach, a parallel multithread reactive TS phase is followed by a distributed *Path Relinking* (PR) phase, i.e., all processes switch from TS to PR simultaneously.

Martins, Ribeiro and Souza [16], proposed a parallel grasp for the Steiner problem in graphs. A Greedy Randomized Adaptive Search Procedure (GRASP) is a meta-heuristic for combinatorial optimization. A GRASP is an iterative process, where each iteration consists of two phases: construction and local search. The construction phase of the algorithm is based on a version of distance network heuristic which is improved by Mehlhorn. Some heuristics are used in order to speedup the local search. For parallelization of GRASP, each slave processor performs a fixed number of GRASP iterations. Once all processors have finished their computations, the best solution is collected by the master processor. The results of computational experiments illustrate the effectiveness of the proposed parallel GRASP procedure for the Steiner problem in graphs.

Verhoeven and Severens proposed sequential and parallel local search methods for the Steiner tree problem based on a novel neighborhood. They claimed their approach is "better" than those known in the literature. Computational results indicated that good speedups could be obtained without loss in solution quality [4].

4. Our Contribution

Having studied all the work explained in the previous section, we were to select a proper algorithm to be parallelized. Among those algorithms having exponential execution time, we concluded that parallelization may not be wise or effective, because as the network scales up and the number of nodes grows, the complexity grows exponentially and therefore the number of required processors would be exponential too. In other words, such algorithms suffer from scalability problem. But among those algorithms having polynomial execution time – and so in the category of

Approximation Algorithms – the only acceptable approximation ratios were 2 – due to ADH and MSTH – , 11/6 and 1.55 – due to Zelikovsky. As Zelikovsky assumes the network graph is complete, the solution may be found only in special cases. Furthermore, although the algorithm runs in $O(\lg^2 n)$ time using $O(n^3)$ processors, this solution is only found with probability at least 1/2 [17]. So the only possible candidates for our purpose are ADH and MSTH. They have been experimentally compared and none of them is proved to be superior of the other in all situations [9]. Shortest path heuristic is not good as its performance ratio is not constant, but bounded by *m*, where *m* is the number of destination vertices.

The ADH algorithm works as follows:

1. Begin with a forest *F* of single node trees, each representing a vertex in *D.*
2. Choose $u \in V$ such that *f(u)* is minimum where

$$f(u) = \min_{S \subseteq F, |S|>1} \frac{1}{|S|-1} \sum_{T \in S} d(u,T).$$

3. Let T_1 and T_2 be two closest trees to *u*.
4. Join T_1 and T_2 by a shortest path through *u*.
5. If $|F|>1$, go to Step 2, else T_{ADH} is the single tree in *F*.

The heuristic has a time complexity of $O(|V|^3)$ since it requires the *all pairs shortest paths* matrix for the graph [9].

At first glance, the algorithm may seem highly parallelizable, but this is not true, since step 5 can not be parallelized. In other words, we face a *sequence* of actions, each of which consists of finding the two candidate trees to be joined and joining these trees. This imposes a limit of $O(n)$ as a lower bound for execution time of any parallel algorithm, which is not acceptable where there exist algorithms with polylogarithmic execution time, so the only remained candidate for the purpose of parallelizing would be the MSTH.

In order to select a good architecture, we noted that this architecture would probably have $O(n^3)$ processors, resulting to $O(\lg^2 n)$ algorithm execution time. This led us to select *three dimensional mesh of trees* (3dMOT) which is a powerful scalable network among several other parallel architectures. 3dMOT consists of a cube of n^3 nodes in which each row of length *n* comprises leaves of a complete binary tree in each of three dimensions [14]. Furthermore, computing the MST of an *n* node graph can be performed on a *two dimensional mesh of trees* which is a subgraph of 3dMOT efficiently in $O(\lg^2 n)$ [14].

Now that we have defined the problem and specified the architecture, we are ready to dive in the problem more formally.

4.1. Problem Formulation and Basic Notation

For the purpose of routing, a communication network may be modeled as a connected, weighted, undirected graph *G = (V, E, w)*, where *V* is the set of vertices representing the nodes in the network, *E* the set of edges representing the links in the network, and *w(e)* is the nonnegative weight associated to the edge *e*, ($\forall e \in E$). When the primary goals are to achieve maximum utilization of the network resources and to ensure load balancing, these weights are assigned to edges to represent the cost of using a link [9]. From now on, we use *n* to represent $|V|$, the total number of nodes in the network. We also define $d(i, j)$ to be the distance – length of the shortest path between – nodes *i* and *j*.

The Steiner problem in graphs asks for the minimum weight tree subgraph interconnecting a subset *D* of vertices – called terminals – in such a graph.

Three Dimensional Mesh of trees, the architecture used to solve this problem, is an $n \times n \times n$ array of processors each element of which is a *leaf processor* which we denote by a triple (*i, j, k*) indicating the position of that processor in dimension 1, 2 and 3 respectively. On each of the dimensions of such an array in 3dMOT, there exists a two dimensional array of trees whose leaves are the array elements. For the sake of simplicity, we represent these trees with triples like (*i, j,* -), in which the dash sign represents the tree's dimension and the two numbers represent the position of the tree among other trees in the same dimension. We've assumed that the roots of trees are linked together among dimensions; this won't add much to the computational power or cost of the network as discussed in [14], but only is to simplify the notation and algorithm explanation.

4.2. The Algorithm

The *minimum spanning tree heuristic* (MSTH), developed by Kou et al., finds potential Steiner vertices assuming that they will likely be on the shortest paths between the destination vertices. The heuristic proceeds as follows:

1. Construct a complete graph $G' = (D, E')$ where $Cost_{G'}(u, v)$ is the length of the shortest path from *u* to *v* in *G*.
2. Construct a minimum spanning tree T' for G'.
3. Construct a subgraph G'' of *G* with all vertices of T'.

4. Construct a minimum spanning tree T'' for G''.
5. Remove successively any pendants which are non-destination leaf vertices in T'' to form a solution, T_{MSTH}.

As presented in [9], the heuristic has a time complexity of $O(|D||V|^2)$.

The parallel algorithm we have proposed to be run on a 3dMOT is the following:

1. Compute *all pairs shortest paths* (APSP) matrix for the graph, i.e. the length of shortest path between all pairs whether in *D* or not in *D*. This will result in formation of a complete graph in which especially between each pair of vertices of *D*, there exist an edge with the weight of minimum cost path linking them, we call these edges as *virtual edges*.
2. Compute MST of the graph G', taking into account only the vertices in *D*.
3. Construct G'' by marking all the edges in G which are contained in at least one of the virtual edges of T'.
4. Construct MST of G'', T'' keeping track of whether each edge joins two required components – and therefore should be preserved – or not.

To explain how to implement these steps on a 3dMOT, we first show how to compute APSP matrix. We simply use the matrix multiplication and squaring technique discussed in [14]. This is accomplished by powering the weight matrix of the graph to *n*-1, in which matrix multiplication and powering is defined like this:

$$C = AB$$
$$c_{ij} = \min_{1 \le l \le n} \{a_{il} + b_{lj}\}$$

$$a_{ij}^{(k)} = \min_{1 \le l \le n} \{a_{il} + a_{lj}^{(k-1)}\}$$

At most $\lceil \log(n-1) \rceil$ multiplications are needed since it suffices to compute A^M where *M* is the smallest power of 2 greater than or equal to *n*-1. Each multiplication takes $2\log n + 1$ steps when entering the matrix to roots of dimension 1 and 3 trees and getting the result at the roots of dimension 2 trees after sending each element to all leaves, multiplying the values in each leaf an summing up the values in leaves of dimension 2 trees. The total time required for matrix multiplication in this way is $2\log^2 n + 3\log n + 1$ steps [14].

To compute the MST as explained in [14] on two dimensional mesh of trees, we need to put the adjacency matrix of G' on leaves of one 2-dimensional mesh of trees in the array. This can be accomplished by sending each value from roots of dimension 3 trees to say its uppermost leaf. We assume that node information – i.e. membership in *D* or not – is present at the roots of trees in this level. Then assuming the *membership in D* to work as enable signal for the processors, the edges of minimum spanning tree can be determined in $O(\log^2 n)$ steps, using the algorithm presented in [14].

To construct G'', we find the nodes belonging to it. We use the leaf (*i, j, k*) to indicate if the node *j* is on the shortest path from node *i* to node *k*. We note that if this is the case, we would have the following:

$$d(i,k) = d(i,j) + d(j,k)$$

If the shortest paths are unique, this is of course correct. Otherwise, then this may cause to construct a supergraph of G'', however this would not affect cost of the result, because we remove the additional nodes and edges at the next steps, when removing the pendants at last.

To do this, each tree root when the tree is in the form (-, *j, k*), (*i*, -, *k*), or (*i, j*, -), sends the distance to all its descendent leaves. Each leaf (*i, j, k*) receiving the required values from the tree roots in each of three dimensions, can verify that it is on a virtual edge connecting *i* and *k* and in MST of G'. To verify that the edge (*i, k*) is in the MST of G', the computed values for existing edges of the MST matrix in the leaf level are sent to corresponding roots of dimension 2 trees. Once we have these values at roots, we send them downward through the tree edges to all leaves with the form (*i, x, k*) that are about to verify the existence of edge (*i, k*). Whenever a node is proved to be on a shortest path contained in the MST, this information should be sent to the corresponding root node of the 2-dimensional array on which the shortest paths matrix is computed – i.e. the uppermost leaf plain. This is a reduction operation since all the information in nodes (*x, j,* y) should be "summarized" by an OR operation in the root of the (-, *j, n*) dimension 1 tree. First, all the leaf processors having an address of the form (*x, j, y*) send their information to their dimension 1 tree root. The result is sent from root to the rightmost leaf node which has an address of the form (*0, j, x*). These leaves which are all descendents of the tree (*0, j, x*) sum up their values by sending them upward to the root of this tree. The final result is then sent to the (*0, j, n*) leaf which in turn sends it to the (-, *j, n*) tree root. These values are sent downward through dimension 1 and 2 trees to the upper plate leaves. Only the edges that receive such enable signals from both of their corresponding trees are considered to be in G''.

The remaining work is to compute the minimum spanning tree of the result graph and removing the pendants. One may think that this is easily accomplished by running the MST on the upper plate and then

removing the degree 1 non-destination nodes. But it's immediately seen that removing these pendants may result in formation of other pendants and this can make us perform a "sequence" of updates, leading the algorithm running time to be of $O(n)$. Instead of doing this, we propose a solution in which we keep track of the needed non-destination nodes during the formation of the MST. For this purpose we need to distinguish between the connected components containing at least one destination node and the components without any destination node. We'll refer to the former as D-components and the latter as S-components. We also call destination nodes as D-node and non-destination nodes as S-nodes. The nodes that are proved to be needed for constructing the Steiner tree will be referred to as N-nodes (needed) and the other nodes will be called U-nodes(unneeded). Furthermore, we assume that among the information contained in each node, we have one flag to indicate whether or not the node is a pendant of its component. This flag is initialized to true (is pendant) for S-nodes and false for D-nodes. We should update this flag when the tree components are being connected to each other. When merging two trees, we may encounter one of these:

1. One of the components is S-component and the other a D-component,
2. Both the components are S-component,
3. Both the components are D-component

In the first and second case no flag needs to be updated. In the third case, if we indicate two ends of the newly added edge by *s* and s', only the flag of the nodes belonging to the paths from *s* and s' to their nearest N-nodes in each component should be changed from true to false, meaning that the node is now needed to construct the tree, because as it's apparent in Figure 2, it would be on the only path from one N-node to another. If any of two edge ends is an N-node, no change in the flags of that component is needed.

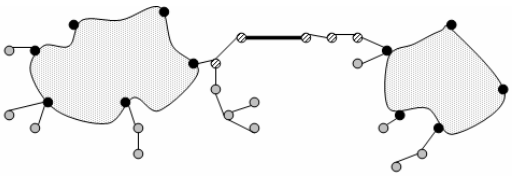

Figure 2. Two components are to be linked together. Black nodes indicate N-node and gray nodes are U-nodes. The thick edge is the newly added edge. It can be seen that hatched nodes are U-node before connecting the new edge, but they should be changed to N-nodes. Also it's obvious that the state of other nodes needn't change.

To do so, we note that when we connect to components while constructing the MST, we should be able to distinguish between D-components and S-components. This is easily accomplished by giving priority to D-nodes to be leader of a component. In this way, if a component is a D-component, its leader would be a D-node and vice versa.

To determine which nodes are there on the path connecting one end of an edge to its nearest N-node, we maintain in the leaf matrix the path-lengths between any pair of nodes in the current forest and represent it by d_{MST}. Then we can verify this fact by checking if the following equality holds:

$$d_{MST}(s,n_s) = d_{MST}(s,i) + d_{MST}(i,n_s)$$

Here, *s* can represent any of U-nodes at one end of the new edge and n_s is the nearest N-node to *s* in its component. Since there is only one path in tree between any pair of nodes, to the update path lengths, when connecting two components, we only need to compute path lengths between pairs of nodes that are not in the same component. This value for two arbitrary nodes n_1 and n_2 as indicated in Figure 3 would be:

$$d_{MST}(n_1,n_2) = d_{MST}(n_1,s_1) + w(s_1,s_2) + d_{MST}(n_2,s_2)$$

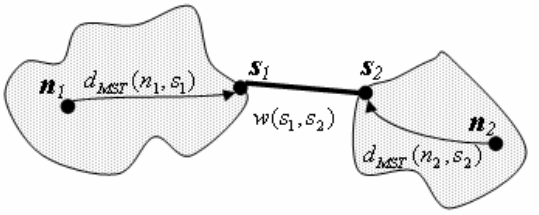

Figure 3. Computing the length of newly established paths when connecting two components.

To do the above on a 2 dimensional mesh of trees (2dMOT) whose leaves are located at the upper plate, we run a modified from of the ordinary MST algorithm on 2dMOT. Whenever we want to connect two supernodes, as soon that the linking edge information was received by the leader (i.e. the weight, edge ends, and leader information of the two ends), it can verify if the two components are both D-component. If this is true, then each node *i* in that component must have $d_{MST}(s,d_s)$, $d_{MST}(s,i)$ and $d_{MST}(i,d_s)$ available to verify if it should be set to an N-node.

At first, the leader sends identity of s downward through row trees. The cell in column *s* that receives this message sends a "calculate nearest N-node" message to its column tree root. The root in turn sends a message to all its leaves to send their d_{MST} if they connect *s* to an N-node. The minimum of these values is calculated as they move upward through the column tree. This minimum value (which embeds the identity of *s* and d_s) is then sent to the leader root.

We show how a leader can send a message to all its component members and will use this abstraction later several times. The leader of any component sends its identity and the message to all its descendent leaves through row trees and each node sends identity of its leader to its column through column trees. In each column *i*, the leaf that has received a message from a leader that is the same as the leader identity received through the column tree, sends the message upward through the column tree to the root node.

The leader sends the identity of *s* and d_s to all its component members. Each member sends its identity along with the identity of *s* and d_s received from leader to all its column tree leaves and the proper leaves send $d_{MST}(s,i)$ and $d_{MST}(i,d_s)$ upward to the root. Now that root has all of the three values $d_{MST}(s,d_s)$, $d_{MST}(s,i)$ and $d_{MST}(i,d_s)$ available, can set its flag properly. This is the process through which we can update the N-nodes.

In order to update path lengths, once again when we want to connect two supernodes, as soon that the linking edge information was received by the leader, it sends identity of s_1 and s_2 along with $w(s_1,s_2)$ to all its component members. Each node, having received these values sends the identity of s_1 and $w(s_1,s_2)$ downward through the column trees to all leaves. The proper leaf then sends $d_{MST}(n_1,s_1)$ to the root which in turn broadcasts this value through row and column trees to its corresponding row and column. In this way, now the leaf (n_1, n_2) having received $d_{MST}(n_1,s_1)$, $d_{MST}(n_2,s_2)$, and $w(s_1,s_2)$ can calculate the path length.

All these actions described here are done when we want to connect two components and The time needed for each of these tasks is $O(\lg n)$ which is of the same order of the time steps needed to do during ordinary MST construction. This means that in the overall time of MST algorithm, only a constant factor may change and the time order is still $O(\lg^2 n)$.

4.3. Time Complexity Analysis

We performed some analysis of time complexity during algorithm explanation, here we will summarize all the statements claimed before for each step of the algorithm and calculate the overall time complexity:

1. Construct $G'=(D,E')$, which is a complete graph in which $Cost_{G'}(u,v)$ is the length of the shortest path from *u* to *v* in *G*.

   This step was performed by repeatedly squaring the matrix, which takes $2\log^2 n+3\log n+1$ steps to be completed.
2. Construct a minimum spanning tree T' for G'.

   This step was accomplished by running the minimum spanning tree algorithm on 2-dimensional mesh of trees, which is known to be possible in $O(\lg^2 n)$ steps [14].
3. Construct a subgraph G'' of *G* with all vertices of T'.

   We did this by sending some values from leaves to roots and vice versa for a few times. This can be done in $O(\lg n)$ steps.
4. Construct a minimum spanning tree T'' for G''.
5. Remove successively any pendants which are non-destination leaf vertices in T'' to form a solution, T_{MSTH}.

   Steps 4 and 5 were done together and as we discussed when explaining the algorithm, this takes $O(\lg^2 n)$ time.

By summing up the values and orders computed above, we conclude that the total time needed for algorithm execution is $O(\lg^2 n)$.

5. Conclusion

We presented a parallel algorithm on a 3-dimensional mesh of trees to solve SPG based on minimum spanning tree heuristic. This algorithm can be further improved because only in limited steps we do need the whole network, and in the remaining time of the algorithm we just perform the task on a 2-dimensional mesh. To make the algorithm better we may think of pipelining multiple problems through the network.

The application of this fast algorithm may be in multimedia applications such as E-learning and video conferencing where the members of the multicast group are not permanent, but change dynamically. Each time the membership of one member is changed, the algorithm can be run and the new multicast tree is generated immediately. Although a lot of processors are needed for this task, the computational power of such processors is limited because they should do simple work. In comparison to the only centralized parallel implementation we've encountered during our study – which computes the solution with a probability at least 0.5, this algorithm may be better in the sense that it certainly generates the solution.

6. References

[1] E. Alba and J. F. Chicano, "Evolutionary Algorithms in Telecommunications", in *IEEE Mediterranean Electrotechnical Conference*, May 2006, pp. 795-798. URL:http://neo.lcc.uma.es/staff/francis/pdf/melecon06.pdf.

[2] M. P. Bastos and C. C. Ribeiro, "Reactive Tabu Search with Path Relinking for the Steiner Problem in Graphs", In *Proceedings of the Third Metaheuristics International Conference*, 1999, pp. 31-36.
URL:http://citeseer.ist.psu.edu/bastos99reactive.html.

[3] T. G. Crainic, M. Gendreau, and J. Potvin, "Parallel Tabu Search", *Parallel Metaheuristics*, E. Alba (Ed.), John Wiley \& Sons, 2005.
URL:http://www.iro.umontreal.ca/~gendron/Pisa/References/Meta/Crainic05c.pdf.

[4] T. G. Crainic and N. Hail, "Parallel Meta-heuristics Applications", *Parallel Metaheuristics*, E. Alba (Ed.), John Wiley \& Sons, 2005.
URL:http://www.iro.umontreal.ca/~gendron/Pisa/References/Meta/Crainic05b.pdf.

[5] D. E. Drake and S. Hougardy, "On Approximation Algorithms for the Terminal Steiner Tree Problem", *Information Processing Letters, vol. 89, Number 1,* January 2004, pp. 15-18.
URL:http://www.sciencedirect.com/.

[6] M. Demange, J. Monnot, and V. Th. Paschos, "Differential Approximation Results for the Steiner Tree Problem", *Applied Mathematics Letters, vol. 16, Issue 5*, July 2003, pp. 733-739.
URL:http://www.sciencedirect.com.

[7] L. M. A. Drummond, E. Uchoa, A. D. Goncalves, J. M.N. Silva, M. C.P. Santos, and M. C. S. de Castro, "A grid-enabled distributed branch-and-bound algorithm with application on the Steiner Problem in graphs", *Parallel Computing, vol. 32, Issue 9*, October 2006, pp. 629-642.
URL:http://www.sciencedirect.com/.

[8] H. Esbensen and P. Mazumder, "A Genetic Algorithm for the Steiner Problem in a graph", *In Proceedings of the European Design and Test* Conference, 1994, pp. 402-406.
URL:http://citeseer.ist.psu.edu/185181.html.

[9] A. Ghanwani, "Neural and delay based heuristics for the Steiner problem in networks", *European Journal of Operational Research, vol. 108, Issue 2,* 16 July 1998, pp. 241-265.
URL:http://www.sciencedirect.com/.

[10] P. Guitart, "A Faster Implementation of Zelikovsky's 11/6-Approximation Algorithm for the Steiner Problem in Graphs", *Electronic Notes in Discrete Mathematics, vol. 10,* November 2001, pp. 133-136.
URL:http://www.sciencedirect.com.

[11] L. Gatani, G. Lo Re, and S. Gaglio, "An efficient distributed algorithm for generating and updating multicast trees", *Parallel Computing, vol. 32, Issues 11-12*, December 2006, pp. 777-793.
URL:http://www.sciencedirect.com.

[12] G. Kulkarni, "A Tabu Search Algorithm for the Steiner Tree Problem", M.Sc. Thesis, North Carolina State University, 2002.
URL:http://www.lib.ncsu.edu/theses/available/etd-09032002-115648/unrestricted/etd.pdf.

[13] Z. Kun, W. Heng, and L. Feng-Yu, "Distributed multicast routing for delay and delay variation-bounded Steiner tree using simulated annealing", *Computer Communications, vol. 28, Issue 11*, 5 July 2005, pp. 1356-1370.
URL:http://www.sciencedirect.com/.

[14] F.T. Leighton, *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes, Morgan Kaufmann*, 1992.

[15] C.P. Low and Y.J. Lee, "Distributed multicast routing, with end-to-end delay and delay variation constraints", *Computer Communications, vol. 23, Issue 9*, 15 April 2000, pp. 848-862.
URL:http://www.sciencedirect.com.

[16] S. L. Martins, C. C. Ribeiro, and M. C. Souza, "A Parallel GRASP for the Steiner Problem in Graphs", *Lecture Notes In Computer Science; vol. 1457*, 1998, pp. 285-297.
URL:http://citeseer.ist.psu.edu/martins98parallel.html.

[17] H. J. Pr¨omel and A. Steger, "A New Approximation Algorithm for the Steiner Tree Problem with Performance Ratio 5/3", *Journal of Algorithms, vol. 36, Issue 1,* July 2000, pp. 89-101. URL:http://www.sciencedirect.com.

[18] C. C. Ribeiro and M. C. De Souza, "Improved Tabu Search for the Steiner Problem in Graphs", Working paper, Catholic University of Rio de Janeiro, Department of Computer Science, 1997.
URL:http://citeseer.ist.psu.edu/47337.html.

[19] G. Robins and A. Zelikovsky, "Improved Steiner Tree Approximation in Graphs", *In Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms*, San Francisco, California, United States, 2000, pp. 770-779.
URL:http://citeseer.ist.psu.edu/robins00improved.html.

[20] V. V. Vazirani, "Recent results on approximating the Steiner tree problem and its generalizations", *Theoretical Computer Science, vol. 235, Issue 1*, 17 March 2000, pp. 205-216.
URL:http://www.sciencedirect.com.