J Inf Process Syst, Vol.13, No.1, pp.104~113, February 2017
https://doi.org/10.3745/JIPS.03.0061

ISSN 1976-913X (Print)
ISSN 2092-805X (Electronic)

JOURNAL OF INFORMATION PROCESSING SYSTEMS JIPS

Parallel and Sequential Implementation to Minimize the Time for Data Transmission Using Steiner Trees

V. Anand* and N. Sairam*

Abstract

In this paper, we present an approach to transmit data from the source to the destination through a minimal path (least-cost path) in a computer network of n nodes. The motivation behind our approach is to address the problem of finding a minimal path between the source and destination. From the work we have studied, we found that a Steiner tree with bounded Steiner vertices offers a good solution. A novel algorithm to construct a Steiner tree with vertices and bounded Steiner vertices is proposed in this paper. The algorithm finds a path from each source to each destination at a minimum cost and minimum number of Steiner vertices. We propose both the sequential and parallel versions. We also conducted a comparative study of sequential and parallel versions based on time complexity, which proved that parallel implementation is more efficient than sequential.

Keywords

Least-Cost Path, Non-terminal Nodes, Parallel, Sequential, Steiner Vertices, Terminals, Time Complexity

1. Introduction

The Steiner tree problem attracts considerable interest because of its applications in various areas. A Steiner tree problem ascertains a Steiner minimum tree, that is, a Steiner tree of the least length. A Steiner tree may be comprised of Steiner points. The length of a Steiner tree is measured by adding the length of all the edges. The Steiner tree problem aims at constructing a network N, which interconnects a set of t terminals with the following characteristics:

1. T may contain nodes other than the given terminals.
2. T minimizes a given cost function.
3. The given cost function guarantees that T can be assumed to be a minimum spanning tree on its nodes (for a given metric).

The first characteristic gives rise to a problem in topology, which is that there is an exponential increase in the number of different topologies to be taken. A control has to be enforced on the number of extra nodes. That is, the first characteristic should be replaced as explained below.

T may contain up to k nodes, other than the given terminals, where k is a given positive integer. The

Manuscript received April 13, 2015; accepted July 16,2015.
Corresponding Author: V. Anand (anandwithah@gmail.com)
* School of Computing, SASTRA University, Tirumalaisamudram, Thanjavur, Tamil Nadu, India ({anandwithah, indsai}@gmail.com)

Steiner tree problem has applications in network routing and wireless communication networks.

We attempted to construct a minimal-Steiner tree. Our approach constructs a minimal-Steiner tree, which includes non-terminal nodes in the shortest path, only if necessary. Our approach does not assign equal weight for all edges. Each edge is randomly assigned a weight based on exponential distribution. These features make our approach capable of overcoming the problem of irregular topologies. We tested our approach both sequentially and in parallel.

This paper is laid out as follows: Section 2 presents the related work, Section 3 discusses the proposed approach, Section 4 presents the algorithm for the sequential version and its analysis, Section 5 presents the algorithm for the parallel version and its analysis, Section 6 presents the results, and the conclusion is given in Section 7.

2. Related Work

Two approximation algorithms have been proposed for Steiner connected dominating sets [1]. One algorithm is used for finding a small dominating set and choosing a terminal as the representative for each vertex in that set, and the other algorithm is for selecting a core vertex and connecting the terminals that have been left out.

The approach proposed in [2], presents an $O(n^2)$ time approximation algorithm for the minimum rectilinear Steiner tree. They showed that the algorithm has a high approximation ratio.

The work proposed in [3] is a technique of linking a collection of terminals by a Steiner tree in a randomized setup. The work proposed a 2-stage stochastic optimization with recourse model for the Steiner tree problem. The model is for an undirected graph $G=(V,E)$, in which V represents vertices and E represents edges, C_e refers to the cost of the edge and a collection of terminals, where $g = \{t_1, t_2, \ldots, t_k\}$, and it finds a subset E', which consists of the least cost edges that links all of the terminals.

The author in [4], proposed a new approach, where he offered algorithms for approximating the directed Steiner tree problem, set cover problem, and generalized Steiner tree problem. He proposed the new primal-dual algorithm, which maintains the double key in the inner of the double feasible region, unlike traditional primal-twofold algorithms. This new approach avoids many arcs in the result, thereby achieving a lesser-cost result. This approach is the inner-point version of the primal-twofold, which performs better than the actual primal-dual method.

The approach proposed in [5] contributes polynomial time approximation schemes for the ϵ-dense Steiner tree problems. Upon comparison with the total number of terminals, if the number of terminal sets is small, their approach proved that the ratio $1+O((\sum_i \log|S_i|)/(\sum_i |S_i|))$, where, S_i is the terminal sets and it becomes less.

The study done in [6], proved that a graph G that is thought to be symmetric has a weight or cost function, where, weight(i, j) = weight(j, i) for all edges (i, j), which belongs to E, where "weight" is the edge weight function. In these cases, the solution of a directed least cost Steiner tree problem and a least cost Steiner tree for an equal undirected weighted graph are alike.

An approximation algorithm based on linear programming uses a randomized rounding technique [7], which is done iteratively. The algorithm is implemented as a directed-component cut reduction for a Steiner tree limited to k. The value of the related variable in the optimum fractional solution is taken. Then, the sampling is done for one of the components with the probability relative to that value and

contract it. The process is repeated, which yields the sampled components with a least-cost terminal spanning tree in the graph. Since the integrality gap is a maximum of 1.55, this algorithm answers the question of: "Is there any LP-relaxation for a Steiner tree with an integrality gap less than 2?".

A generalised method to solve the *k*-Steiner tree problem [8] deals with Steiner minimum trees, which may contain up to *k*-Steiner points for a given constant *k*. This approach generalizes the 1-Steiner tree problem proposed in [9], which gives an optimal solution within $O(n^2)$ time. But, the approach proposed in [8] offers the solution in $O(n^{2k})$ time.

The authors of [10] have shown that the problem of reducing the length of the lengthy edge in a Steiner tree with atmost *k*-Steiner vertices is NP-hard and cannot be approximated within a factor 2-ε, for any $\varepsilon > 0$, unless $P = NP$. This proposed algorithm performs one of the following operations for the given edge $e_i \in E$: (a) It either constructs a *k*-ST in G with bottleneck at most $2d(e_i)$, or (b) it returns the information that G_i does not contain a *k*-ST.

A hybrid algorithm was proposed in [11]. The algorithm focused on solving the Steiner tree problem and was built on the modified intelligent water drops algorithm and learning automata. The intelligent water drops algorithm is efficient because of its global search and quick convergence abilities. To improve its performance, learning automata was used to select the parameters of the intelligent water drops algorithm. This hybrid algorithm performs better than conventional heuristic algorithms and other algorithms with quick convergence. For time-constrained problems, the proposed Extended Intelligent Water Drops (EIWD) algorithm offers the benefit of quick convergence.

The algorithm presented in [12] is for computing a min-cost tree *S*, which spans all terminals. The aim was to minimize the total power consumption of nodes. The power of a node *v* was computed as edges of the *S* incident to *v* are taken, calculate the cost of these edges and find the maximum of it. The min-power version of the problem is suitable for wireless applications.

The approach proposed in [13] concentrates on arbitrary cases of the Steiner tree problem. Taking the performance analysis of the algorithms on typical instances, the focus was to harmonize the typical worst case study of algorithms with them. For a given random graph, the work done in [14] constructed minimum Steiner trees with the weights of all of the edges being the same, which were equal to 1. In a complete graph K_n, with *n* vertices, this approach sought to find the least weight of a Steiner tree. The weights of the edges were selected independently from a distribution *X*.

3. Proposed Approach

This paper aims at constructing a minimal-Steiner tree. There are many algorithms available for constructing a minimal-Steiner tree. However, our approach does it with the improvement of randomly assigning weights and with a control over the number of extra nodes (non-terminals). Our approach also constructs a weighted Steiner tree, with the edges having random weights.

As stated by the author of [15], a weighted Steiner tree can be defined as an undirected network $G=(n,e,\delta,C)$, where δ is a mapping from the collection of vertices to the collection of reals called "weights," and let *T* be a non-empty set, $T \subseteq n$ called terminals. The weighted Steiner tree problem is defined as:

Find $S_G(T)$→ a subnetwork of *G* such that ∀ pair of terminals ∃ a path

$$\min \sum_{n_i \in s_G(T)} w_i + \sum_{(n_i, n_j) \in s_G(T)} C_{ij} \quad (1)$$

where, W is the weight of edges and C is the cost of nodes. A Steiner tree is a collection of terminals and non-terminals (Steiner points), which is shown in Fig. 1.

Fig. 1 shows a collection of terminals (v_i) and non-terminals (u_j), For terminals (v_i), "i" varies from 1 to 7 and for non-terminals (u_j) "j" varies from 1 to 3.

However, the issue is that the maximum number of non-terminals leads to different topologies. Our approach focuses on the shortest path with the least cost. If the shortest path needs some non-terminals in the paths to destinations, then only the non-terminals will be included in the paths towards destinations. This solves the problem of different (irregular) topologies.

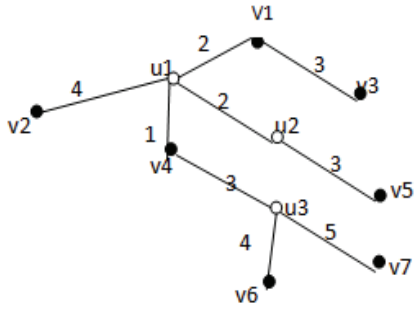

Fig. 1. A Steiner tree with 7 terminals and 3 non-terminals (Steiner points).

This approach obtains the number of terminals and non-terminals as input and the cost matrix is formed. Each edge is randomly assigned a weight based on exponential distribution. Using the matrix, our algorithm finds the shortest path between vertices. A minimal tree is constructed with all terminal nodes. Finally, our internodes procedure finds the intermediate non-terminal nodes in the path between the source and destinations and, thus, a minimal-Steiner tree is constructed.

Our approach is implemented in two ways—sequentially and in parallel. Sequential implementation is done with C language and for parallel implementation, openMP in RedHat Linux Ubuntu release is used.

4. Algorithms for a Sequential Version

4.1 Finding the Minimal Steiner Tree

In a minimal Steiner tree problem, the vertices are separated into terminal and non-terminals. The solution or the output must have the terminals. The weights of all edges are added to calculate the weight or cost of a Steiner tree. To find the least-cost Steiner tree, the tree might encompass some Steiner points (non-terminals). Our approach produces the results at a minimum cost by including the non-terminals only on demand. The following procedure generates a cost matrix with a set of nodes by

calling routines to find the shortest path and finds the minimal tree to connect terminal nodes. The algorithms are presented below.

To find a minimal-Steiner tree

Step 1: Give the input for the number of terminal and non-terminal nodes.
Step 2: The cost matrix is formed by randomly assigning weights based on exponential distribution.
Step 3: Call the shortest path to find the minimal path between vertices.
Step 4: Call the minimal tree to find the minimal tree that connects the terminal nodes.
Step 5: FOR i = 1 TO 2*T steps of 2
call the internodes to find the intermediate nodes
between sttree[i] and sttree[i+1]
END FOR

4.2 Finding the Shortest Path

The algorithm presented here finds the shortest path from a chosen source to a given destination. It partitions vertices into two distinct groups, a group of not-placed vertices and a group of placed vertices. To begin with, all vertices are not-placed, and the algorithm ends once all vertices are in the placed set. A vertex is moved from the not-placed *set* to the placed set, once its shortest distance from the source has been found. This algorithm is so powerful that it finds all of the shortest paths from the source to all of the destinations.

To find the shortest path

Step 1: FOR k= 1 TO n
Step 2: FOR i = 1 TO n
Step 3: FOR j = 1 TO n
Step 4: IF ($arr[i][j] > arr[i][k] + arr[k][j]$)
$arr[i][j] = arr[i][k] + arr[k][j]$;
$path[i][j]=k$;
END IF
END FOR
END FOR
END FOR

4.3 Finding the Minimal Spanning Tree

Given a graph G, the algorithm produces a minimum spanning tree of G. Our approach finds the minimal spanning tree in this way: a tree is an acyclic graph. The algorithm starts with an empty graph and adds edges one at a time, so that it is acyclic. The resulting graph is always a subset of some minimum spanning tree.

To find the minimal spanning tree

```
Step 1: v[0]=TRUE;
Step 2: FOR i = 2 TO n
            v[i]=FALSE;
        END FOR
Step 3: sttree[1]=1, s=0,k=2;
Step 4: WHILE (k<n)
        min=999;
        FOR i = 1 TO n
          FOR j = 2 TO n
            IF(v[i]==TRUE&&v[j]==FALSE&&c[i][j]<min)
                min=c[i][j];
                v1=I;
                v2=j;
            END IF
          END FOR
        END FOR
        mincost=mincost+min;
        v[v2]=TRUE;
        k++,sttree[s++]=v1,sttree[s++]=v2;
      END WHILE
```

4.4 Finding the Intermediate Nodes

In the algorithm to find the minimal-Steiner tree, internodes is called in Step 5. This procedure is called to find the intermediate nodes between the source and destination, if there are any.

To find the intermediate nodes

```
internodes(path[][MAX],arr[][MAX],r1,c1)
Step 1: i=r1,j=c1, node[nno++]=r1
Step 2: Print (r1);
Step 3: WHILE(path[i][j]!=-1)
            PRINT (path[i][j])
            i=path[i][j];
        END WHILE
Step 4: PRINT(c1)
```

4.5 Analysis of the Computation-Sequential Version

The procedure to find the minimal spanning tree has a time complexity of $O(T^2)$, where T is the number of terminals. This is extremely fast compared to the exact rectilinear Steiner minimal tree (RSMT) algorithm. The time complexity of the algorithm to get the least-cost path is $O(E \log n)$, where E refers to edges and n refers to vertices. The number of edges is determined from the value of T, and for each edge internodes is called. The time complexity of internodes is computed based on the count of

Steiner points (non-terminal nodes) between the terminals, and the time complexity is $O(T^*n)$, where T refers to the terminals and n refers to all the nodes.

5. Algorithm for the Parallel Version

The parallel version of our program was designed and executed using OpenMP. The procedure for finding the intermediate nodes was parallelized. Each path from the source to the destination runs on a separate thread and each thread consumes a different time for execution.

5.1 To Find the Minimal-Steiner Tree

Step 1: Give the input for the number of terminal nodes and non-terminal nodes.
Step 2: The cost matrix is formed by randomly assigning weights based on exponential distribution.
Step 3: Call the shortest path to find the minimal path between vertices.
Step 4: Call the minimal tree to find the minimal tree connecting terminal nodes.
Step 5: Parallel section begins
    Parallel FOR i = 1 TO 2*T steps of 2
        call the internode to find the intermediate
        nodes between *sttree*[i] and *sttree*[i+1]
    END the parallel FOR
Parallel section ends

To prove that the tree formed is a minimum Steiner spanning tree, we have provided the Lemma 1. For proof, we used the assumption that ∃ is a minimum spanning tree SP_T for $u,v \in n$ such that:

$$\max(P_{SP_T}(u,v)) < \max(P_{ST_T}(u,v)). \quad (2)$$

By rule of contradiction, it has been proved that ST_T is the minimum Steiner spanning tree.

Let us consider the following notations for the proof given in Lemma 1. Let us denote the minimum Steiner spanning tree as ST_T, and $P_{ST_T}(u,v)$ be a unique path between u and v in the minimum Steiner spanning tree ST_T. Let us assume that the maximum length of an edge in $P_{ST_T}(u,v)$ is the max ($P_{ST_T}(u,v)$).

Lemma 1. The algorithm finds the minimum Steiner spanning tree ST_T.

Proof: Let us assume that ∃ is a minimum spanning tree SP_T for $u,v \in$ n, such that:

$$\max(P_{SP_T}(u,v)) < \max(P_{ST_T}(u,v)).$$

Therefore, ∃ is an edge (u',v') of $P_{ST_T}(u,v)$ that is longer than the edge of $P_{SP_T}(u,v)$. Let us remove (u',v') from ST_T. Therefore, ST_T will be partitioned into two components that are connected. If we add an edge (u",v") to ST_T , it will make the graph connected by joining the connected components. This will create a Steiner spanning tree ST_T' with:

$$\begin{aligned} \text{length}(ST_T') &= \text{length}(ST_T) - \text{length}((u',v')) + \text{length}((u'',v'')) \\ &\leq \text{length}(ST_T) - \max(P_{SP_T}(u,v)) + \max(P_{ST_T'}(u,v)) \\ &< \text{length}(ST_T) \end{aligned} \quad (3)$$

A contradiction to our definition of the minimum Steiner spanning tree. Therefore, our assumption is wrong and ST_T is the minimum.

5.2 Analysis of the Computation-Parallel Version

Our proposed approach calculates the total execution time using the computation time and communication time, as given below:

$$\text{Total execution time} = \text{computation time} + \text{communication time}. \quad (4)$$

In the parallel version, parallelization is done for finding the intermediate non-terminal nodes. The time complexity for finding the intermediate non-terminal nodes is $O(n)$, where n refers to all the nodes. The communication time is computed as:

$$\text{Communication time} = n * (t_{start} + m_i * t_{data}) + \alpha \quad (5)$$

where, 't_{start}' is the start time, 'm_i' is the amount of data sent by i^{th} thread to the master thread, 't_{data}' is the time taken for sending the data of each 'm_i', 'n' is the total number of threads. And 'α' is the time for communication between the threads.

6. Results and Discussion

Apart from proving that the tree formed is a minimum Steiner spanning tree, we have analyzed the parallel and sequential versions and found that the computation time of the parallel version is less than the computation time of the sequential version. The results are illustrated in Table 1. We arrived at the results based on different values of n and T.

Table 1. Computation time in parallel version vs. computation time in sequential version

	Total no. of nodes (n)	**No. of terminal nodes (T)**	**Computation time (s)**	**Communication time (s)**
Parallel version	100	75	0.013	1.075
	70	65	0.011	0.670
	50	30	0.006	0.298
	50	20	0.004	0.284
	25	10	0.007	0.079
Sequential version	100	75	1.107	-
	70	65	0.692	-
	50	30	0.347	-
	50	20	0.292	-
	25	10	0.093	-

Fig. 2 shows a graph, plotted with a number of terminal nodes along the x-axis and the computation time along the y-axis. The graph illustrates a comparison of the computation times for the parallel and sequential versions.

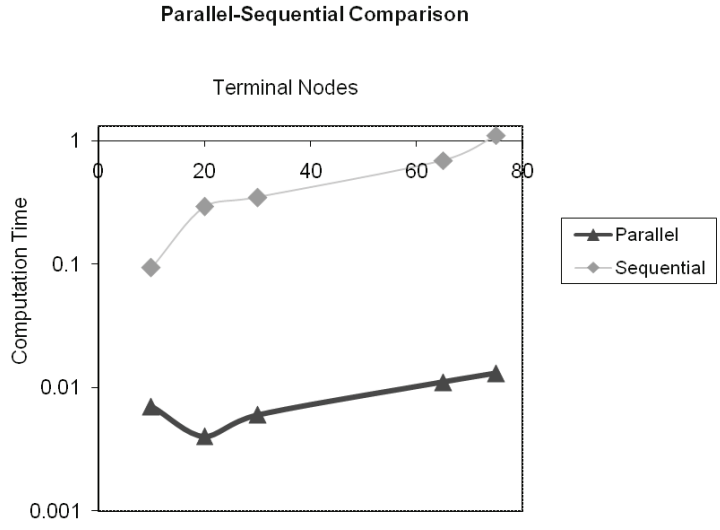

Fig. 2. Graph for parallel-sequential comparison.

7. Conclusion

The proposed approach consumes a number of terminals as input and constructs a minimal Steiner tree. It builds paths from the source to destinations with regard to multicasting. Our approach achieves cost reduction by including the non-terminals only on demand. Since the weights are randomly assigned to edges based on exponential distribution, the proposed approach is effective for a large number of nodes, which results in multiple paths between the source and destinations. This allows our approach to offer reliable transmission. We conducted our experiments on few terminals and on large numbers also. The experimental results of the sequential and parallel versions were compared and reveal that parallel execution works faster than the sequential version in finding the paths with the minimum cost. Beyond the results we have proved, we found that randomly assigning weights using exponential distribution appears to be a good model for multicasting, but, not a good model with regard to the topology of the network. This seems to be an open problem for using different distributions on Steiner tree-based networks with different topologies. A lot of research on Steiner trees in relation to network communication can still be done.

References

[1] Y. F. Wu, Y. L. Xu, and G. L. Chen, "Approximation algorithms for Steiner connected dominating set," *Journal of Computer Science and Technology*, vol. 20, no. 5, pp. 713-716, 2005.

[2] J. MA, B. Yang, and S. Ma, "A practical algorithm for the minimum rectilinear Steiner tree," *Journal of Computer Science and Technology*, vol. 15, no. 1, pp. 96-99, 2000.
[3] A. Gupta and M. Pal, "Stochastic Steiner trees without a root," in *Automata, Languages, and Programming.* Heidelberg: Springer, 2005, pp. 1051-1063.
[4] V. Melkonian, "New primal-dual algorithms for Steiner tree problem," *Journal of Computers and Operations Research*, vol. 34, no. 7, pp. 2147-2167, 2007.
[5] M. Hauptmann, "On the approximability of dense Steiner problems," *Journal of Discrete Algorithms*, vol. 21, pp. 41-51, 2013.
[6] D. Skorin-Kapov and J. Skorin-Kapov, "A note on Steiner tree games," *Journal of Networks*, vol. 59, no. 2, pp. 215-225, 2012.
[7] J. Byrka, F. Grandoni, T. Rothvoß, and L. Sanita, "An improved LP-based approximation for Steiner tree," in *Proceedings of the 42nd ACM Symposium on Theory of Computing (STOC)*, Cambridge, MA, 2010, pp. 583-592.
[8] M. Brazil, C. J. Ras, K. J. Swanepoel, and D. A. Thomas, "Generalised k-Steiner tree problems in normed planes," in *Computing Research Repository*, 2011 [Online]. Available: http://arxiv.org/pdf/1111.1464v1.pdf.
[9] G. Georgakopoulos and C. H. Papadimitriou, "The 1-Steiner tree problem," *Journal of Algorithms*, vol. 8, no. 1, pp. 122-130, 1987.
[10] A. K. Abu-Affash, P. Carmi, and M. J. Katz, "Bottleneck Steiner tree with bounded number of Steiner vertices," in *Proceedings of 23rd Canadian Conference on Computational Geometry*, Toronto, 2011, pp. 355-366.
[11] S. Noferesti and H. Shah-Hosseini, "A hybrid algorithm for solving Steiner tree problem," *International Journal of Computer Applications*, vol. 41, no. 5, pp. 14-20, 2012.
[12] F. Grandoni, "On min-power Steiner tree," in *Proceedings of 20th Annual European Symposium on Algorithms (ESA)*, Ljubljana, Slovenia, 2012, pp. 527-538.
[13] B. Bollobas, D. Gamarnik, O. Riordan, and B. Sudakov, "On the value of a random minimum weight Steiner tree," *Combinatorica*, vol. 24, no. 2, pp. 187-207, 2004.
[14] L. Kucera, A. Marchetti-Spaccamela, M. Protasi, and M. Talamo, "Near optimal algorithms for finding minimum Steiner trees on random graphs," in *Mathematical Foundations of Computer Science*. Heidelberg: Springer, 1986, pp. 501-511.
[15] A. Segev, "The node-weighted Steiner tree problem," *Networks*, vol. 17, no.1, pp. 1-17, 1987.

V. Anand

He has received the MCA degree from A.V.C. College, Bharathidasan University and M.Phil degree in Computer Science from Manonmaniam Sundaranar University, Tirunelveli, Tamil Nadu, India. He is doing Ph.D. in SASTRA University. Currently, he is an Assistant Professor at SASTRA University, Thanjavur. His research interests include computer networks, parallel processing and database systems.

N. Sairam

He has received the M.Tech. degree in Computer Science and Engineering from SASTRA University, Thanjavur. He has received the Ph.D. degree in Computer Science from SASTRA University, Thanjavur. Currently, he is a Professor at SASTRA University, Thanjavur. His research interests include distributed algorithms, parallel processing, data mining and genetic algorithms.