Efficient Steiner Tree Construction Based on Spanning Graphs

Hai Zhou

***Abstract*—The Steiner Minimal Tree (SMT) problem is a very important problem in very large scale integrated computer-aided design. Given n points on a plane, an SMT connects these points through some extra points (called Steiner points) to achieve a minimal total length. Even though there exist many heuristic algorithms for this problem, they have either poor performances or expensive running time. This paper records an implementation of an efficient SMT algorithm that has a worst case running time of $O(n \log n)$ and a performance close to that of the Iterated 1-Steiner algorithm. The algorithm efficiently combines Borah *et al.*'s edge substitute concept with Zhou *et al.*'s spanning graph. Extensive experimental studies are conducted to compare it with other programs.**

***Index Terms*—Graph algorithms, routing, Steiner tree.**

I. Introduction

THE Steiner Minimal Tree (SMT) problem has wide applications in very large scale integrated (VLSI) computer-aided design. Given n points on a plane, an SMT connects these points through some extra points (called Steiner points) to achieve a minimal total length. An SMT is generally used in initial topology creation for noncritical nets in physical synthesis. For timing critical nets, minimization of wire length is generally not enough. However, since most nets are noncritical in a design and an SMT gives the most desirable route of such a net, it is often used as an accurate estimation of congestion and wire length during floorplanning and placement. This implies that a Steiner tree algorithm will be invoked millions of times. On the other hand, there exist many large preroutes in modern VLSI design. The preroutes are generally modeled as large sets of points, thus increasing the input sizes of the Steiner tree problem. Since the SMT is a problem that will be computed millions of times and many of them will have very large input sizes, highly efficient solutions with good performance are desired.

Because of its importance, there is much previous work to solve the SMT problem. These algorithms can be grouped into two classes: exact algorithms and heuristic algorithms. Since SMT is NP-hard, any exact algorithm is expected to have an exponential worst case running time. However, two prominent achievements must be noted in this direction. One is the `GeoSteiner` algorithm and implementation by Warme *et al.* [14], [15], which is the current fastest exact solution to the problem. The other is a polynomial time approximation scheme (PTAS) by Arora [1], which is mainly of theoretical importance. Since exact algorithms have long running time, especially on large input sizes, much more previous efforts were put on heuristic algorithms. Many of them generate a Steiner tree by improving on a minimal spanning tree topology [6], since it was proved that a minimal spanning tree is a 3/2 approximation of an SMT [7]. However, since the backbones are restricted to the minimal spanning tree topology in these approaches, there is a reported limit on the improvement ratios over the minimal spanning trees. The iterated 1-Steiner algorithm by Kahng and Robins [10] is an early approach to deviate from that restriction and an improved implementation [4] is a champion among such programs in public domain. However, the implementation in [10] has a running time of $O(n^4 \log n)$ and the implementation in [4] has a running time of $O(n^3)$. A much more efficient approach was later proposed by Borah *et al.* [2]. In their approach, a spanning tree is iteratively improved by connecting a point to an edge and deleting the longest edge on the created circuit. Their algorithm and implementation had a worst case running time of $\Theta(n^2)$, even though an alternative $O(n \log n)$ implementation was also proposed. Since the backbone is no longer restricted to the minimal spanning-tree topology, its performance was reported to be similar to the iterated 1-Steiner algorithm [2]. A recent effort in this direction is a new heuristic by Mandoiu *et al.* [11], which is based on a 3/2 approximation algorithm of the metric Steiner tree problem on quasi-bipartite graphs [12]. It performs slightly better than the iterated 1-Steiner algorithm, but its running time is also slightly longer than the iterated 1-Steiner algorithm (with the empty rectangle test [11] used).

Our objective in this paper is to implement an SMT heuristic that is much faster than the iterated 1-Steiner algorithm and has similar performance. To achieve that goal, we select the edge-substitution approach of Borah *et al.* [2] as the basis, and enhance it with the spanning graph of Zhou *et al.* [17] and other improvements. The implemented algorithm runs in $O(n \log n)$ time and takes $O(n)$ storage, without large hidden constant. Another advantage of the algorithm is that it is easy to be implemented. Extensive experimental studies are conducted to compare it with other SMT programs and its advantages are demonstrated. Even though we only focus on rectilinear SMT (where distances are measured with rectilinear metric), the same idea can also be used for Euclidean or octilinear Steiner trees.

Manuscript received June 10, 2003; revised August 18, 2003. This work was supported in part by the National Science Foundation under Grant CCR-0238484. This paper was recommended by Associate Editor J. Lillis.

The author is with the Department of Electrical and Computer Engineering, Northwestern University, Evanston, IL 60208 USA (e-mail: haizhou@ece.nwu.edu).

Digital Object Identifier 10.1109/TCAD.2004.826557

The rest of the paper is organized as follows. In Section II, the edge-substitution approach of Borah *et al.* [2] and the spanning graph of Zhou *et al.* [17] will be reviewed as the basis of our program. Then, in Section III, we will show how the spanning graph can be used both to generate the initial spanning tree and to find the point-edge pairs for edge substitution. Section IV will give experimental results.

II. Background

A. Borah et al.'s Edge-Substitution Approach

As illustrated by the example in Fig. 1, Borah *et al.*'s algorithm [2] for the rectilinear SMT works as follows. It starts with a minimal spanning tree and then iteratively considers connecting a point (for example p in Fig. 1) to a nearby edge [for example, (a, b)] and deleting the longest edge ((b, c)) on the circuit thus formed.

A straightforward implementation by Borah *et al.* [2] used Prim's algorithm [3] to generate the initial minimal spanning tree in $O(n^2)$ time, and considered all possible point-edge pairs in the given tree, whose number is n^2. To find the longest edge on the circuit formed by connecting each point-edge pair, a depth-first search was conducted on the minimal spanning tree starting from every edge. The longest edge on the path from the starting edge to the current point is, thus, the longest edge on the circuit formed by connecting the current point to the starting edge. Since each depth-first search from one edge takes $\Theta(n)$ time, the total time for all edges is $\Theta(n^2)$. To keep the running time within $O(n^2)$, only one point-edge pair with the maximal gain was kept for each edge, and the $O(n)$ pairs were sorted according to nonincreasing gains. Each point-edge pair was then connected (with proper deletion of the longest edge in the circuit) if the two involved edges had not been changed.

However, the straightforward implementation, as described in [2], was not totally correct. The problem came from the way the longest edge for a point-edge pair was found. To see how that happened, consider a minimal spanning tree as shown in Fig. 2. From each edge in the tree, a depth-first search will be used to find the longest edge on the path to every point. Thus, when starting from edge (a, b) in Fig. 2, the longest edge to c could be (d, e) or (f, g). Suppose we pick the closest to the point, it is (f, g). Then, when we start from edge (a', b'), similarly, the longest edge to c' could be (d, e) or (f, g). Using the same criteria will give us (d, e). Therefore, when c is connected to (a, b), edge (f, g) will be deleted. Then when c' is connected to (a', b'), edge (d, e) will be deleted. This will give a dangling edge (e, f) and a loop from c to c'.

Besides the straightforward implementation, Borah *et al.* [2] also discussed a possible way to make an algorithm of $O(n \log n)$ running time. This was based on an observation that not every point needs to be considered with every edge. For example, in Fig. 1, point d does not need to be considered with edge (a, b), since they are blocked by edge (c, e). Using the geometrical blockage information, a point only needs to be considered with visible edges from its position. The number of point-edge pairs is thus reduced to $O(n)$. Unfortunately, this requires a geometrical sweepline algorithm to generate visible point-edge pairs. Tarjan's offline least common ancestor algorithm [3] must be used to compute the longest edges on created

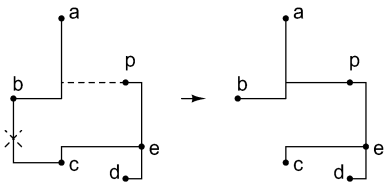

Fig. 1. Edge substitution by Borah *et al.*.

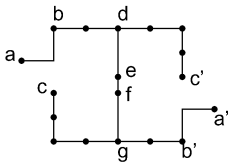

Fig. 2. Problem in the straightforward implementation of Borah *et al.*.

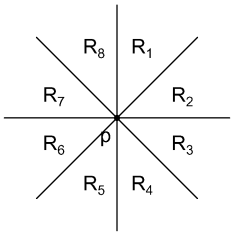

Fig. 3. Point p needs to be connected to at most one point in each region.

circuits by point-edge connections. To keep the initial minimal spanning-tree generation within $O(n \log n)$ time, Hwang's algorithm for minimal spanning-tree construction [8] was suggested. As we can see, since each stage of this suggested approach involves a separate algorithm and some of them are very complicated, it is much more complicated than the straightforward implementation and was never implemented.

B. Zhou et al.'s Spanning Graph

Zhou *et al.* [17] introduced the spanning graph as an intermediate step for the minimal spanning-tree construction.

Definition 1: Given a set of points on the plane, a spanning graph is a graph on the points that contains at least one minimal spanning tree.

The number of edges in the graph is called the cardinality of the graph and they presented an efficient $O(n \log n)$ algorithm to construct a spanning graph of cardinality $O(n)$.

Similar to Yao [16] and Guibas and Stolfi [5], from each point p, the plane is divided into eight octal regions as shown in Fig. 3. It can be proved that if rectilinear distance (that is, the distance between two points (x_1, y_1) and (x_2, y_2) is given by $|x_1 - x_2| + |y_1 - y_2|$) is used, then the distance between any two points in one region is always smaller than the maximal distance from them to p. Based on the cycle property of a minimal spanning tree, that is, the longest edge on any circuit should not be included in any minimal spanning tree, this means that only the closest point to p in each region needs to be connected to p. Considering all given points, the connections will form a spanning graph of cardinality $O(n)$.

A sweepline algorithm was designed by Zhou *et al.* [17] to efficiently construct the above spanning graph. It has a worst

```
Algorithm Rectilinear Spanning Graph (RSG)
  for (i = 0; i < 2; i++) {
    if (i == 0) sort points according to x + y;
    else sort points according to x − y;
    A[1] = A[2] = ∅;
    for each point p in the order {
      find points in A[1], A[2] such that p is in their
        R_{2i+1} and R_{2i+2} regions, respectively;
      connect p with points in each subset;
      delete the subsets from A[1], A[2], respectively;
      add p to A[1], A[2];
    }
  }
```

Fig. 4. Rectilinear spanning-graph algorithm.

case running time of $O(n \log n)$ and is much simpler than the divide-and-conquer algorithm of Guibas and Stolfi [5]. Even though Guibas and Stolfi's algorithm has the same worst case runtime, the recursive approach is more complex and may take $O(n \log n)$ storage. The pseudocode of Zhou *et al.* is presented in Fig. 4 and works as follows. In order to find the closest points in regions R_1 and R_2 for all points, the points will be sorted in nondecreasing order of $x + y$. In this way, after each point p is swept, the first point seen in its R_1 or R_2 region will be the closest point in that region. To keep the swept points waiting for closest points in R_1 and R_2 regions, two active sets A_1 and A_2 are used. When a new point is swept, we need to search A_1 to find points with the new point in their R_1 region, and to search A_2 to find points with the new point in their R_2 region. Then, edges are added from the new point to these points. After the found points are deleted from A_1 and A_2, the new point is added to them, since now the new point is swept and is waiting for the closest points. The connections for all points to their closest points in R_3 and R_4 regions are computed in the exact same fashion, except that now points are swept in nondecreasing order of $x - y$. There is no need to consider connections in regions R_5 through R_8, since they have been implied by connections in regions R_1 through R_4.

To achieve $O(n \log n)$ running time, the active sets A_1 and A_2 must be efficiently maintained so that searching, deletion, and insertion each can be done in $O(\log n)$ time. It can be shown that when an active set is used for region R_i, it cannot have two points such that one is in the other's R_i region. This property guarantees that the points in each active set be linearly ordered. Therefore, a balanced search tree could be used to efficiently implement an active set.

III. Steiner Tree Algorithm Based on Spanning Graphs

Since the edge substitution of Borah *et al.* [2] is a simple, yet effective, approach for SMT construction, our algorithm is based on it. Compared with Hwang's $O(n \log n)$ time algorithm [8] they suggested for the initial minimal spanning tree, Zhou *et al.*'s minimal spanning-tree algorithm [17] based on spanning graph is much more efficient and simpler to implement. Furthermore, in their suggestion, a computational geometry algorithm needs to be used to generate the visible relations between points and edges in the tree. However, we find that if a spanning graph is generated, then the geometrical proximity information between points and edges is already embedded in the

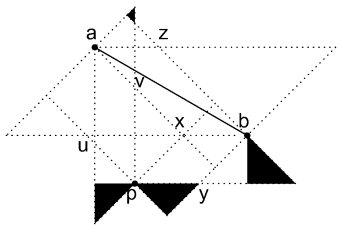

Fig. 5. Point visible to an edge is usually connected to one end point in the spanning graph.

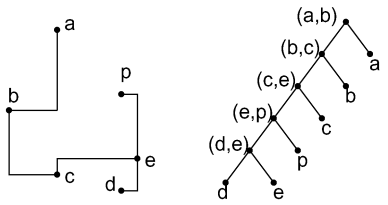

Fig. 6. A minimal spanning tree and its merging binary tree.

```
Algorithm Rectilinear Steiner Tree (RST)
  T = ∅;
  Generate the spanning graph G by RSG algorithm;
  for (each edge (u, v) ∈ G in non-decreasing length) {
    s1 = find_set(u); s2 = find_set(v);
    if (s1 != s2) {
      add (u, v) in tree T;
      for (each neighbor w of u, v in G)
        if (s1 == find_set(w))
          lca_add_query(w, u, (u, v));
        else lca_add_query(w, v, (u, v));
      lca_tree_edge((u, v), s1.edge);
      lca_tree_edge((u, v), s2.edge);
      s = union_set(s1, s2); s.edge = (u, v);
    }
  }
  generate point-edge pairs by lca_answer_queries;
  for (each pair (p, (a, b), (c, d)) in non-increasing positive gains)
    if ((a, b), (c, d) has not been deleted from T) {
      connect p to (a, b) by adding three edges to T;
      delete (a, b), (c, d) from T;
    }
```

Fig. 7. RST algorithm.

spanning graph. Therefore, no extra sweepline routine is needed to compute the edge blockage for point-edge pair generation if the spanning graph is leveraged. This makes the spanning graph a backbone of the whole algorithm. It is first used to generate the initial minimal spanning tree, and then to generate point-edge pairs for tree improvements. This kind of unification happens also in the spanning-tree computation and the longest edge computation for each point-edge pair; using Kruskal's algorithm with disjoint set operations (instead of Prim's algorithm) [3] will unify these two computations.

In order to reduce the number of point-edge pair candidates from $O(n^2)$ to $O(n)$, Borah *et al.* suggested to use the visibility of a point from an edge. That is, only a point visible from an edge can be considered to connect to that edge. However, it requires a sweepline algorithm to find visibility relations between points and edges. In order to skip this complex step, we decide to leverage the geometrical proximity information embedded

TABLE I
COMPARISON WITH OTHER ALGORITHMS I

input size	GeoSteiner		BI1S		BOI		RST	
	improve	time	improve	time	improve	time	improve	time
100	11.440	0.487	10.907	0.633	9.300	0.0267	10.218	0.004
200	11.492	3.557	10.897	4.810	9.192	0.1287	10.869	0.020
300	11.492	12.685	10.931	18.770	9.253	0.2993	10.255	0.041
500	11.525	72.192	-	-	9.274	0.877	10.381	0.084
800	11.343	536.173	-	-	9.284	2.399	10.719	0.156
1000	-	-	-	-	9.367	4.084	10.433	0.186
2000	-	-	-	-	9.326	31.098	10.523	0.381
3000	-	-	-	-	9.390	104.919	10.449	0.771
5000	-	-	-	-	9.356	307.977	10.499	1.330

TABLE II
COMPARISON WITH OTHER ALGORITHMS II

input size	BGA		Borah		Rohe		RST	
	improve	time	improve	time	improve	time	improve	time
Randomly generated testcases								
100	10.272	0.006	10.341	0.004	9.617	0.000	10.218	0.002
500	10.976	0.068	10.778	0.178	10.028	0.010	10.381	0.041
1000	10.979	0.162	10.829	0.689	9.768	0.020	10.433	0.121
5000	11.012	1.695	11.015	25.518	10.139	0.130	10.499	0.980
10000	11.108	4.135	11.101	249.924	10.111	0.310	10.559	2.098
50000	11.120	59.147	-	-	10.109	1.890	10.561	13.029
100000	11.098	161.896	-	-	10.079	4.410	10.514	28.527
500000	-	-	-	-	10.059	27.210	10.527	175.725
VLSI testcases								
337	6.434	0.035	6.503	0.037	5.958	0.010	5.870	0.016
830	3.202	0.070	3.185	0.213	3.102	0.020	2.966	0.033
1944	7.850	0.342	7.772	2.424	6.857	0.040	7.533	0.238
2437	7.965	0.549	7.956	4.502	7.094	0.050	7.595	0.408
2676	8.928	0.623	8.994	3.686	8.067	0.060	8.507	0.463
12052	8.450	4.289	8.465	232.779	7.649	0.300	8.076	2.281
22373	9.848	11.330	9.832	1128.365	8.987	0.570	9.462	4.605
34728	9.046	18.416	9.010	2367.629	8.158	0.900	8.645	5.334

within the spanning graph. Since a point has eight nearest points connected around it, we observed that if a point is visible to an edge, then the point is *usually* connected in the graph to at least one end point. Illustrated in Fig. 5 is a point p and an edge (a, b), where the octal regions of each point are shown by dotted lines. If (p, a) is in the graph, then the region a, u, p, v has no point. Similarly, if (p, b) is in the graph, then the region b, x, p, y has no point. Since (a, b) is also in the graph, the region a, x, b, z has no point, either. On the other hand, when there is no point between p and (a, b), the point p is not connected to either a or b, if and only if there is at least one point in each of the shaded regions. However, we must note that there is no one-to-one correspondence between visibility and connection to the end points. In our algorithm, the spanning graph is used to generate point-edge pair candidates. For each edge in the current tree, all points that are neighbors of either of the end points will be considered to form point-edge pairs with the edge. Since the cardinality of the spanning graph is $O(n)$, the number of possible point-edge pairs generated in this way is also $O(n)$.

When connecting a point to an edge, the longest edge on the formed circuit needs to be deleted. In order to find the corresponding longest edge for each point-edge pair efficiently, we should look at how the spanning tree is formed through Kruskal's algorithm. This algorithm first sorts the edges into nondecreasing lengths and each edge is considered in turn. If the end points of the edge have been connected, then the edge will be excluded from the spanning tree; otherwise, it will be included. The structure of these connecting operations can be represented by a binary tree, where the leaves represent the points and the internal nodes represent the edges. When an edge is included in the spanning tree, a node is created representing the edge and has as its two children the trees representing the two components connected by this edge. To illustrate this, a spanning tree with its representing binary tree are shown in Fig. 6. As we can see, the longest edge between two points is the least common ancestor of the two points in the binary tree. For example, the longest edge between p and b in Fig. 6 is (b, c), which is the least common ancestor of p and b in the binary tree. To find the longest edge on the circuit formed by connecting a point to an edge, we need to find the longest edge between the point and one end point of the edge that are in the same component before connecting the edge. For example, consider the pair p and (a, b), since p and b are in the same component before connecting (a, b), the edge needs to be deleted is the longest between p and b.

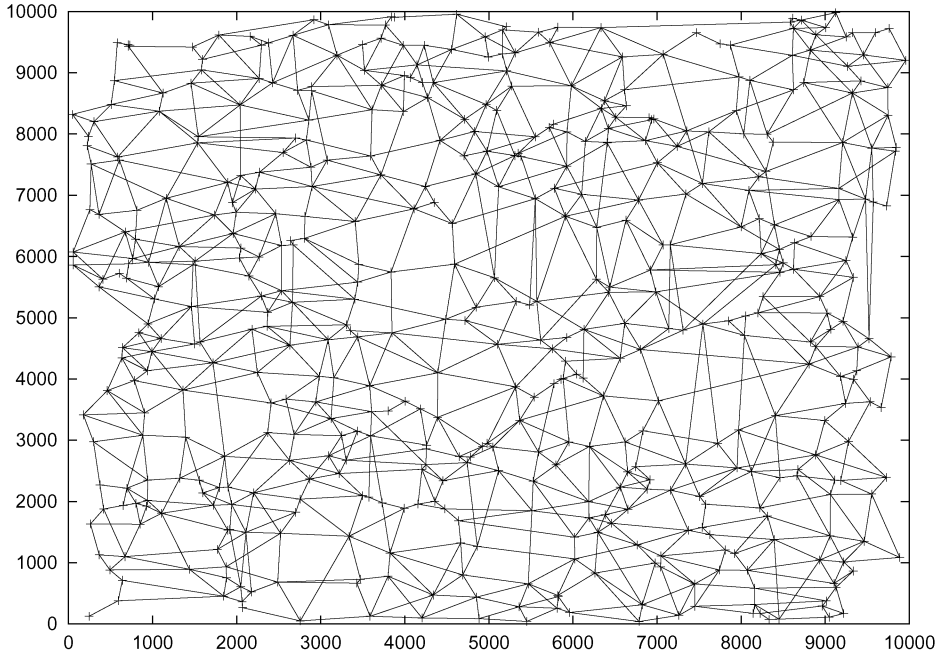

Fig. 8. Spanning graph over 500 points.

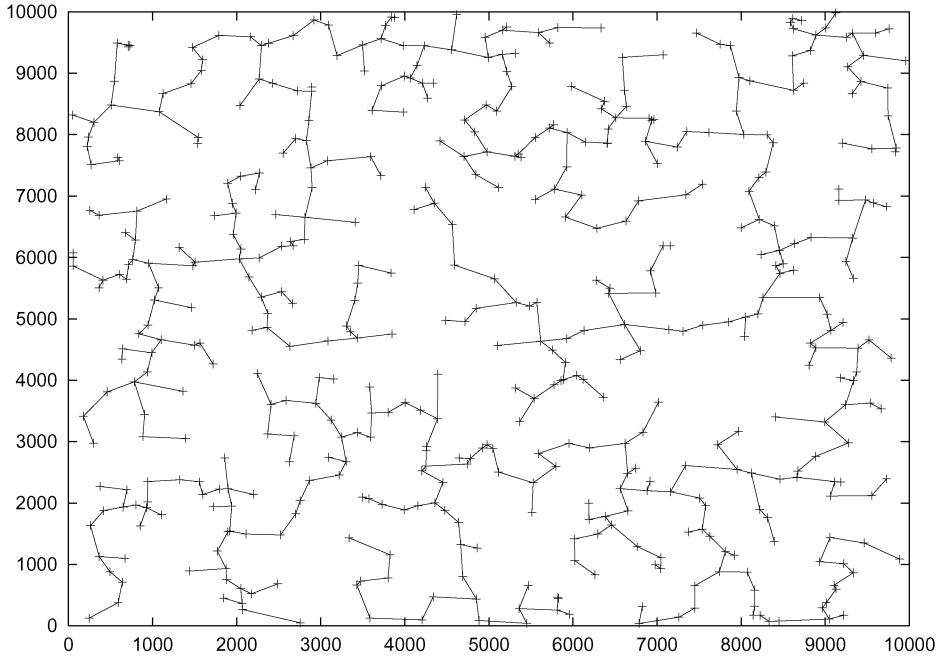

Fig. 9. Minimal spanning tree over 500 points.

Based on the above discussion, the pseudocode of the algorithm can be described in Fig. 7. At the beginning of the algorithm, Zhou *et al.*'s rectilinear spanning-graph algorithm [17] is used to generate the spanning graph G for the given set of points. Then, Kruskal's algorithm is used on the graph to generate a minimal spanning tree. The data structure of disjoint sets [3] is used to merge components and check whether two points are in the same component (the first **for** loop). During this process, the merging binary tree and the queries for least common ancestors of all point-edge pairs are also generated. Here s, $s1$, and $s2$ represent disjoint sets and each records the root of the component in the merging binary tree. For each edge (u, v) adding to T, each neighbor w of either u or v will be considered to connect to (u, v). The longest edge for this pair is the least common ancestor of w, u or w, v depending on which point is in the same component as w. The procedure `lca_add_query` is used to add this query. Connecting the two components by (u, v) will also be recorded in the merging binary tree by the procedure `lca_tree_edge`. After generating the minimal spanning tree, we also have the corresponding merging binary tree and the least common ancestor queries ready. By using Tarjan's offline least common ancestor algorithm [3] (represented by

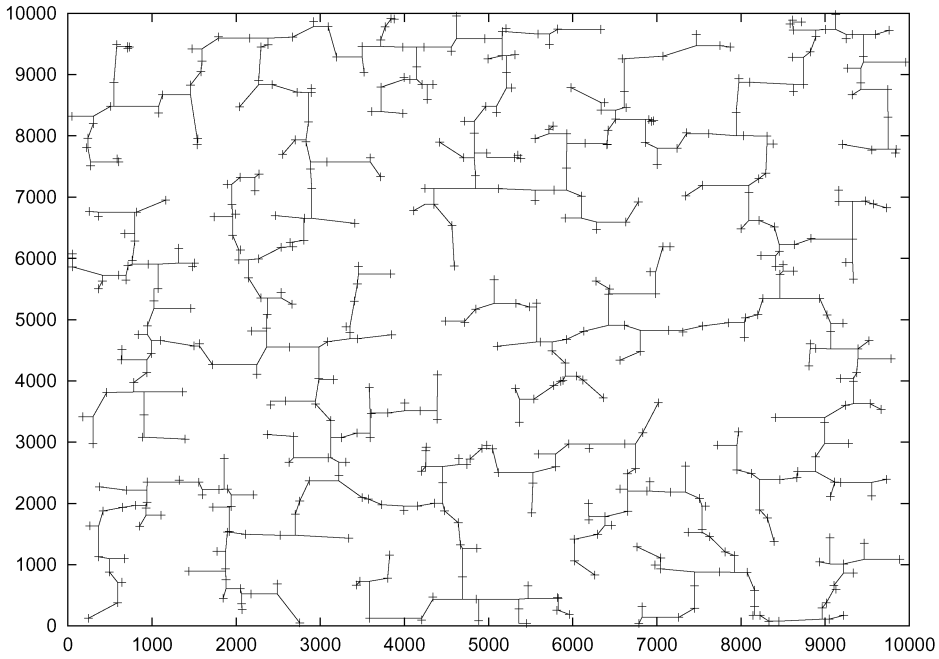

Fig. 10. Steiner tree by RST over 500 points.

lca_answer_queries), we can generate all longest edges for the pairs. With the longest edge for each point-edge pair, the gain of connecting the point to the edge can be calculated. Then, each of the point-to-edge connections will be realized in a non-increasing order of their gains. A connection can only be realized if both the connection edge and deletion edge have not been deleted yet.

The running time of the algorithm is dominated by the spanning-graph generation and edge sorting, which takes $O(n \log n)$ time. Since the number of edges in the spanning graph is $O(n)$, both Kruskal's algorithm and Tarjan's offline least common ancestor algorithm take $O(n\alpha(n))$ time, where $\alpha(n)$ is the inverse of Ackermann's function, which grows extremely slow.

With the unexpected problem of Borah *et al.* discussed in Section II, we should avoid placing the correctness of an algorithm only on our intuitions. Therefore, we formally prove the correctness of our algorithm through the following theorem.

Theorem 1: The RST algorithm only generates a Steiner tree on a given set of points.

Proof: Proof of the theorem reduces to showing the following invariant in the last **for** loop in the algorithm:

For any point-edge pair $[p, (a, b), (c, d)]$, if neither (a, b) nor (c, d) is touched (i.e., deleted), then there must be a path between p and (a, b) that goes through (c, d).

This invariant is true at the beginning of the **for** loop. Suppose it is violated, it must be violated by committing a pair (x, e_1, e_2). And the only possible way is that the deletion edge e_2 is on the path between p and (a, b) and is not (a, b) or (c, d). This implies that (c, d) is an ancestor of e_2 in the binary tree. We claim that the circuit formed by connecting x to e_1 cannot include (c, d). Otherwise, both e_2 and (c, d) are on the path from x to e_1. Since (c, d) is an ancestor of e_2, there is no way for e_2 to be the least common ancestor of x and e_1. Based on the claim, committing (x, e_1, e_2) can change at most a part of the path between p and (a, b) that does not include (c, d), Therefore, after the merging, the path between p and (a, b) still includes (c, d).

Based on the above invariant, the tree property will be kept by each edge substitution. ■

IV. Experimental Results

We implemented the Rectilinear Steiner Tree (RST) algorithm in the C programming language, following exactly the pseudocode in Fig. 7, with the exception that the program starting from the first **for** loop is repeated for five times if there are improvements in the previous iteration.

The first set of experiments were conducted on a Linux system with a 928-MHz Intel Pentium III processor edt–i with 512 M memory. We compared our program (denoted as RST) with other publicly available programs: the exact algorithm GeoSteiner (version 3.1) by Warme *et al.* [14]; the Batched Iterated 1-Steiner (BI1S) by Robins; and the Borah *et al.*'s algorithm implemented by Madden (BOI). All the programs can be found at the Gigascale Silicon Research Center's Achievable Design Bookshelf[1].

In Table I, the results of the first set of experiments are reported. For each input size ranging from 100 to 5000, 30 different test cases are randomly generated through the rand_points program in GeoSteiner. The improvement ratios of a Steiner tree St over its corresponding minimal spanning tree MST is defined as $100 \times (\mathtt{MST} - \mathtt{St})/\mathtt{MST}$. For each input size, we report the average of the improvement ratios and the average running time (in seconds) on each of the programs. As we can see, RST always gives better improvements than BOI

[1]Available at www.gigascale.org/bookshelf.

with less running times. This confirms our belief that there is no large hidden constant in the running time of RST.

The second set of experiments compared RST with Borah's implementation of Borah *et al.*'s algorithm (Borah), Rohe's Prim-based algorithm (Rohe) [13], and Kahng *et al.*'s Batched Greedy Algorithm (BGA) [9]. They were run on a different Linux system with a 2.4-GHz Intel Xeon processor and 2 Gb of memory. Besides the randomly generated test cases, the VLSI industry test cases used in [9] were also used. The results are reported in Table II.

For better understanding of the use of spanning graphs in Steiner tree construction, and the quality of Steiner trees generated by RST, we also plotted the spanning graph, the minimal spanning tree, and the Steiner tree, computed by RST for a randomly generated 500 points, in Figs. 8–10.

V. Conclusion

In summary, we developed an efficient Steiner tree algorithm which is based on Borah *et al.*'s edge substitution approach and Zhou *et al.*'s spanning graph algorithm. The implementation has a running time of $O(n \log n)$ and a storage requirement of $O(n)$, without large hidden constant. Experimental results showed the efficiency of this algorithm.

Acknowledgment

The author would like to thank I. Mandoiu, M. Borah, and A. Rohe for granting access to their implementations.

References

[1] S. Arora, "Polynomial-time approximation schemes for euclidean TSP and other geometric problem," *J. ACM*, vol. 45, no. 5, pp. 753–782, 1998.

[2] M. Borah, R. M. Owens, and M. J. Irwin, "An edge-based heuristic for Steiner routing," *IEEE Trans. Computer-Aided Design*, vol. 13, pp. 1563–1568, Dec. 1994.

[3] T. H. Cormen, C. E. Leiserson, and R. H. Rivest, *Introduction to Algorithms*. Cambridge, MA: MIT Press, 1989.

[4] J. Griffith, G. Robins, J. S. Salowe, and T. Zhang, "Closing the gap: Near-optimal Steiner trees in polynomial time," *IEEE Trans. Computer-Aided Design*, vol. 13, pp. 1351–1365, Nov. 1994.

[5] L. J. Guibas and J. Stolfi, "On computing all north-east nearest neighbors in the L_1 metric," *Inform. Process. Lett.*, vol. 17, no. 4, pp. 219–223, 1983.

[6] J.-M. Ho, G. Vijayan, and C. K. Wong, "New algorithms for the rectilinear Steiner tree problem," *IEEE Trans. Computer-Aided Design*, vol. 9, pp. 185–193, Feb. 1990.

[7] F. K. Hwang, "On Steiner minimal trees with rectilinear distance," *SIAM J. Appl. Math.*, vol. 30, pp. 104–114, 1976.

[8] ——, "An $O(n \log n)$ algorithm for rectilinear minimal spanning trees," *J. ACM*, vol. 26, no. 2, pp. 177–182, 1979.

[9] A. B. Kahng, I. I. Mandoiu, and A. Zelikovsky, "Highly scalable algorithms for rectilinear and octilinear Steiner trees," in *Proc. Asia South Pacific Design Automation Conf.*, 2003, pp. 827–833.

[10] A. B. Kahng and G. Robins, "A new class of iterative Steiner tree heuristics with good performance," *IEEE Trans. Computer-Aided Design*, vol. 11, pp. 893–902, July 1992.

[11] I. I. Mandoiu, V. V. Vazirani, and J. L. Ganley, "A new heuristic for rectilinear Steiner trees," in *Proc. Int. Conf. Computer-Aided Design*, vol. 18, 1999, pp. 1.5.7–1.6.2.

[12] S. Rajagopalan and V. V. Vazirani, "On the bidirected cut relaxation for the metric Steiner tree problem," in *10th ACM–SIAM Symp. Discrete Algorithms*, 1999, pp. 742–751.

[13] A. Rohe, "Sequential and parallel algorithms for local routing," Ph.D. dissertation, Bonn Univ., Bonn, Germany, 2001.

[14] D. M. Warme, P. Winter, and M. Zacharisen. GeoSteiner 3.1 Package. [Online]. Available: ftp.diku.dk/diku/users/martinz/geosteiner-3.1.tar.gz

[15] ——, "Exact algorithms for plane Steiner tree problems: A computational study," Dept. Comput. Sci., Univ. Copenhagen, Copenhagen, Denmark, Tech. Rep. DIKU-TR-98/11, 1998.

[16] A. C.-C. Yao, "On constructing minimum spanning trees in k-dimensional spaces and related problems," *SIAM J. Comput.*, vol. 11, no. 4, pp. 721–736, 1982.

[17] H. Zhou, N. Shenoy, and W. Nicholls, "Efficient spanning tree construction without delaney triangulation," *Inform. Process. Lett.*, vol. 81, no. 5, pp. 271–276, 2002.

Hai Zhou received the B.S. and M.S. degrees in computer science and technology from Tsinghua University, Beijing, China, in 1992 and 1994, respectively, and the Ph.D. degree in computer sciences from the University of Texas, Austin, in 1999.

He is an Assistant Professor of Electrical and Computer Engineering at Northwestern University, Evanston, IL. Before he joined the faculty of Northwestern University, he was with the Advanced Technology Group, Synopsys, Inc. His research interests include very large scale integrated computer-aided design, algorithm design, and formal methods.

Prof. Zhou served on the technical program committees of the ACM International Symposium on Physical Design and the IEEE International Conference on Computer-Aided Design. He was the recipient of a CAREER Award from the National Science Foundation in 2003.