Spanning Graph-Based Nonrectilinear Steiner Tree Algorithms

Qi Zhu, *Student Member, IEEE*, Hai Zhou, *Senior Member, IEEE*, Tong Jing, *Member, IEEE*, Xian-Long Hong, *Fellow, IEEE*, and Yang Yang

***Abstract*—With advances in fabrication technology of very/ultra large scale integrated circuit (VLSI/ULSI), we must face many new challenges. One of them is the interconnect effects, which may cause longer delay and heavier crosstalk. To solve this problem, many interconnect performance optimization algorithms have been proposed. However, when these algorithms are designed based on rectilinear interconnect architecture, the optimization capability is limited. Therefore, nonrectilinear interconnect architectures become a field of active research in which the octilinear interconnect architecture is the most promising one since it extends from the rectilinear case and greatly shortens the wire length. Meanwhile, an interconnect with less length is helpful to reduce wire capacitance, congestion, and routing area. In an interconnect architecture, the Steiner minimal tree (SMT) construction is one of the key problems. In this paper, we give two practical octilinear Steiner minimal tree (OSMT) construction algorithms based on octilinear spanning graphs (OSGs). The one with edge substitution (OST-E) has a worst-case running time of $O(n \log n)$ and a similar performance as the recent work using batched greedy. The other one with triangle contraction (OST-T) has a small increase in the constant factor of running time and a better performance. These two are the fastest algorithms for octilinear Steiner tree construction so far. Experiments on both industrial and random test cases are conducted to compare with other programs. We also proposed the extension of our algorithms to any λ geometry.**

***Index Terms*—Deep submicron (DSM), interconnect, physical design, routing, Steiner tree, very large scale integration (VLSI).**

I. Introduction

ROUTING plays an important role in physical design. Useful algorithms have been proposed focusing on routability [1], [2], timing issues [3]–[7], coupling effects [8], and rectilinear routing tree construction [9], [10]. Steiner minimal tree (SMT) construction based on any interconnect architecture is one key problem in routing, which connects n given points through extra added points (Steiner points) to achieve a minimal total length. An SMT is usually the preferred wiring for a multiple-pin net and thus often used as an accurate estimation for wire length. It will be computed hundreds of thousand times during routing. Meanwhile, it may be computed with very large input size for increased emphasis on design for test and nets with pre-routes. Therefore, the SMT problem definitely deserves good performances and highly efficient solutions.

As the foundation of octilinear interconnect, an octilinear Steiner minimal tree (OSMT) connects given points in the octilinear plane, which allows 45° diagonal interconnections in addition to traditional horizontal and vertical orientations. Studies about octilinear Steiner trees have been made since the 1990s. The approach of [11] uses a heuristic based on a minimal spanning tree (MST) to construct a Steiner tree. It has a $O(\lambda n \log n)$ running time, but the nonrectilinear results were not mentioned. In [12], geometrical characteristics of octilinear Steiner trees were studied, especially on three-point and four-point problems, and an iterative discretized 1-Steiner algorithm was proposed. The implementation requires an $O(n^4 \log n)$ running time. It was mentioned that if the technique in [14] is used, it is possible to reduce the running time to $O(n^3)$. Optimal octilinear Steiner tree constructions for three or four points have been studied also in [13], and an $O(n \log n)$ heuristic based on Delaunay triangulation was proposed. However, it only considered local substitutions, which do not change the general topology of the original minimal spanning tree. Recently, there have been more interests [15]–[19] in octilinear routing due to its advantages in wire length and new technical support [18]. A graph-based interconnect optimization algorithm was proposed in [15]. This GRATS-tree algorithm has a running time of $O(n^2)$ per improvement pass. It got 17% wire length improvement over rectilinear minimal spanning trees (RMSTs) on average. In [17], Chiang *et al.* proposed an algorithm to construct an isomorphic octilinear Steiner tree from a rectilinear Steiner tree. In [19], an efficient approach was recently proposed by Kahng *et al.*. In their approach, a spanning tree is iteratively improved by contracting empty triangles [20]. Their implementation utilized the batched method introduced in [21] and gave an $O(n \log^2 n)$ heuristic with good solution qualities. Besides these heuristic approaches, exact octilinear Steiner tree algorithms have also been studied [22]. However, any exact algorithm is expected to have an

Manuscript received March 14, 2004; revised August 29, 2004. This work was supported in part by the NSFC (China) under Grant 60373012, the NSF (USA) under Grant CCR-0238484, the Specialized Research Fund for the Doctoral Program of Higher Education of China under Grant 20020003008, the Hi-Tech Research and Development (863) Program of China under Grant 2004AA1Z1050, and the Key Faculty Support Program of Tsinghua University under Grant [2002]4. A preliminary version of this work has been published in the IEEE/ACM Asian–Pacific Design Automation Conference (ASP-DAC), Yokohama, Japan, 2004, pp. 687–690. This paper was recommended by Associate Editor M. D. F. Wong.

Q. Zhu was with the Computer Science and Technology Department, Tsinghua University, Beijing 100084, China. He is now with the Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, CA 94720 USA (e-mail: zhuqi99@mails.tsinghua.edu.cn).

T. Jing, X.-L. Hong, and Y. Yang are with the Computer Science and Technology Department, Tsinghua University, Beijing 100084, China (e-mail: zhuqi@eecs.berkeley.edu; jingtong@tsinghua.edu.cn; hxl-dcs@tsinghua.edu.cn; yycs99@mails.tsinghua.edu.cn).

H. Zhou is with the Electrical and Computer Engineering Department, Northwestern University, Evanston, IL 60208-3118 USA (e-mail: haizhou@ece.northwestern.edu).

Digital Object Identifier 10.1109/TCAD.2005.850862

exponential worst-case running time, which is not practical for large input sizes.

The main contributions of this paper are two efficient octilinear Steiner minimal tree algorithms, which are faster than previous algorithms while having similar performance as the best heuristics, such as the batched greedy algorithm [19]. The first algorithm (OST-E) derives its efficiency from combining the edge substitution approach of Borah *et al.* [23] and spanning graphs similar to Zhou *et al.* [24]. The other one (OST-T) combines the triple contraction [20] and spanning graphs. Both approaches run in $O(n \log n)$ time and take $O(n)$ storage. The former one has a smaller constant factor in running time, while the latter one has a better performance. Another advantage is that they are easy to implement since all stages of the algorithms are integrated together, rather than different stages use different algorithms. Furthermore, because they are graph based, they can be extended to any λ geometry easily.

The remainder of this paper is organized as follows. In Section II, the spanning graphs of Zhou *et al.*, the edge substitution approach of Borah *et al.*, and the triple contraction approach of Zelikovsky will be reviewed. In Section III, we will show how to define spanning graphs on the octilinear plane and give an efficient algorithm for their construction. Then, in Section IV, the two algorithms of octilinear Steiner tree construction will be presented. In Section V, we will discuss the extension of our algorithms to Steiner trees in any λ geometry. The last two sections will give experimental results and concluding remarks. A preliminary version of this research has been published in the Asian–Pacific Design Automation Conference (ASP-DAC) [30] to cover a partial work. This paper presents progresses in λ-geometry extension and includes detailed analysis.

II. Preliminaries

A. Zhou et al.'s Spanning Graphs

Our algorithms construct an octilinear Steiner tree from a minimal spanning tree, so the first step is constructing an octilinear minimal spanning tree efficiently. Using Prim's or Kruskal's algorithms on the complete graph will take at least $\Omega(n^2)$ time. Although Delaunay triangulation can be an efficient way to reduce the number of candidate edges, it cannot be defined easily under rectilinear or octilinear case. Therefore, Zhou *et al.* [24] introduced the spanning graph as a base for minimal spanning trees. Given a set of points on the plane, a spanning graph is a graph that contains at least one minimal spanning tree. They also presented an $O(n \log n)$ algorithm to construct a spanning graph with $O(n)$ edges.

With reference to each point s, the plane can be divided into eight octagonal regions as shown in Fig. 1(a). Each region includes only one of its two bounding half lines as shown in Fig. 1(b). It can be proven that each region has the uniqueness property, i.e., for every pair of points $p, q \in R_i$, $\|pq\|_2 < \max(\|sp\|_2, \|sq\|_2)$. Here, $\|pq\|_2$ denotes the distance between p and q in a 2-geometry (rectilinear) plane. Based on the cycle property, i.e., the longest edge on any circuit should not be included in any minimal spanning tree, it can be concluded that it is sufficient to consider only the edge from s to the closest point in each region for minimal spanning tree construction. Considering all given points, such edges will form a spanning graph with $O(n)$ edges.

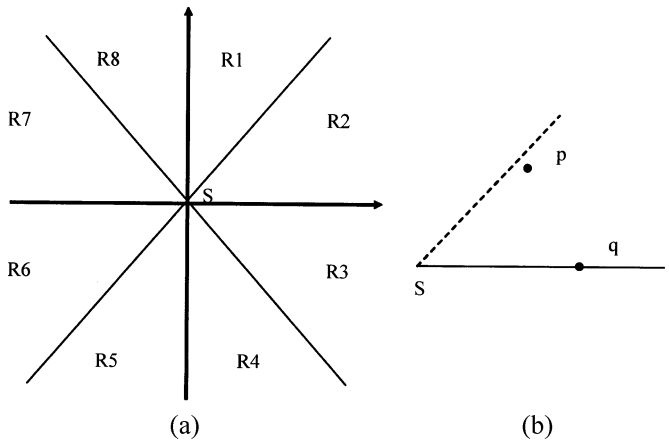


Fig. 1. Octal partition.

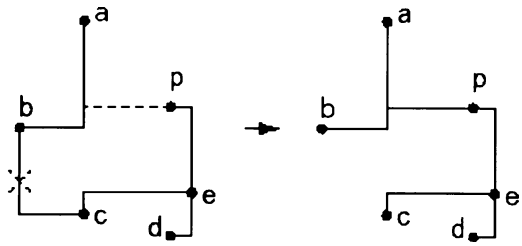


Fig. 2. Edge substitution.

Zhou *et al.* also designed an efficient sweep line algorithm with a worst-case running time of $O(n \log n)$ to construct the spanning graph. We extend this algorithm to the octilinear case, which will be presented in Section III.

B. Borah et al.'s Edge Substitution

The second step of our algorithms is constructing an octilinear Steiner tree from a minimal spanning tree. Our two algorithms are based on edge substitution and triple contraction, respectively.

As illustrated by the example in Fig. 2, Borah *et al.*'s edge substitution algorithm works as follows. It starts with a minimal spanning tree and then iteratively connects a point (for example, p in Fig. 2) to a nearby edge (a, b) and deleting the longest edge (b, c) on the formed cycle.

A straightforward implementation by Borah *et al.* used the Prim algorithm to generate the initial minimal spanning tree in $O(n^2)$ time and considered all possible point–edge pairs, whose number is n^2. A depth-first search was conducted to find the longest edge on the cycle formed by any new connection. This procedure will take $\Theta(n^2)$ time totally. In our approach, we utilize our spanning graph to find the point–edge pairs and the longest edge on a cycle, which will reduce the number of candidates to $O(n)$. This will be presented in Section IV.

C. Zelikovsky's Triple Contraction

Triple contraction by Zelikovsky greedily chose triples of points to improve the minimal spanning tree by connecting them

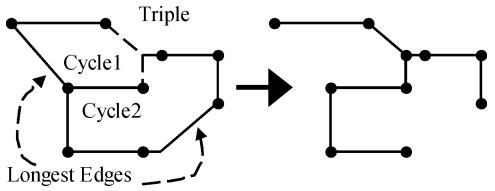


Fig. 3. Triple contraction.

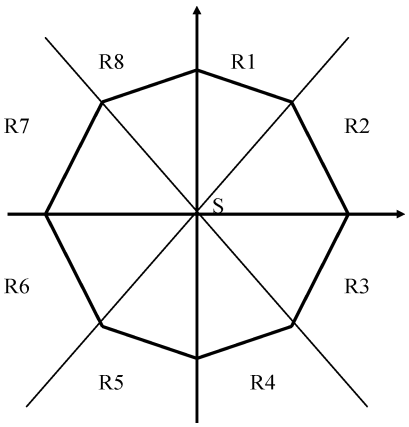


Fig. 4. Equidistant points in octilinear plane.

through Steiner points. As illustrated in Fig. 3, a selected triple is connected, and the two longest edges are deleted in the formed cycles. In fact, this approach considers a superset of the edge substitution and will theoretically have a better performance. Kahng *et al.* utilize this approach to get a $O(n \log^2 n)$ running time heuristic [21]. We will combine triple contraction with spanning graphs to get an $O(n \log n)$ approach.

III. Octilinear Spanning Graphs

To construct a spanning graph in octilinear case, we first consider the contour of equidistant points from point s, as shown in Fig. 4. We can divide the plane into eight regions based on the octagonal contour. In Section V, we will discuss how to extend this idea to any λ geometry.

Theorem 1: Each region R_i in Fig. 4 has the uniqueness property. That is, for any points p, $q \in R_i$, $||pq||_4 < \max(||s||_4, ||sq||_4)$.

Proof: Since regions R_1 through R_8 are equivalent to each other. We only give a proof of R_2 for convenience.

Let x_i denote the x coordinate of i and y_i denote the y coordinate of i $(i = s, p, q)$. And let $||sp||_4$, $||sq||_4$, and $||pq||_4$ denote the distances between points in the octilinear case.

As shown in Fig. 5, we assume points p, q are in region R_2 and without loss of generality, $y_p \geqslant y_q$ (that is, with reference to point q, point p is in the regions R_1, R_2, R_7, and R_8). Furthermore, we assume region R_2 does not include the bounding half line between R_1 and R_2, i.e., the dashed line in Fig. 5.

Case 1: We assume point p is in the region R_8 with reference to point q. Thus

$$\begin{aligned}
||sp||_4 &= x_p - y_p + \sqrt{2}y_p \\
||sq||_4 &= x_q - y_q + \sqrt{2}y_q \\
||pq||_4 &= \sqrt{2}(x_q - x_p) + y_p - y_q - (x_q - x_p).
\end{aligned}$$

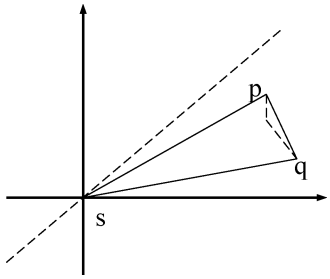


Fig. 5. Case 1.

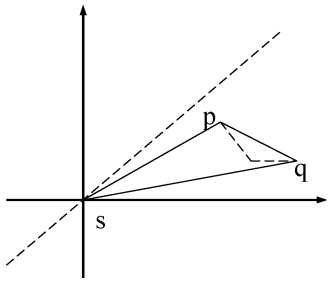


Fig. 6. Case 2.

Therefore

$$\begin{aligned}
||sp||_4 + ||sq||_4 &= x_p + x_q - y_p - y_q + \sqrt{2}y_p + \sqrt{2}y_q \\
&\geqslant x_p(2\sqrt{2}-2)x_q + (\sqrt{2}-1)y_p + (\sqrt{2}-1)y_q \\
&> (\sqrt{2}-2)x_p + (3-\sqrt{2})y_p + (2\sqrt{2}-2)x_q \\
&\quad + (\sqrt{2}-1)y_p - 2y_q \quad (\text{since } x_p > y_p) \\
&= (\sqrt{2}-2)x_p + (2\sqrt{2}-2)x_q + 2y_p - 2y_q \\
&\geqslant (2-2\sqrt{2})x_p + (2\sqrt{2}-2)x_q + 2y_p - 2y_p \\
&= 2\sqrt{2}(x_q - x_p) + 2y_p - 2y_q - 2(x_q - x_p) \\
&= 2 * ||pq||_4.
\end{aligned}$$

We can therefore prove $||pq||_4 < \max(||sp||_4, ||sq||_4)$ in this case.

Case 2: We assume point p is in the region R_7 with reference to point q, as shown in Fig. 6. Thus

$$\begin{aligned}
||sp||_4 &= x_p - y_p + \sqrt{2}y_p; \\
||sq||_4 &= x_q - y_q + \sqrt{2}y_q; \\
||pq||_4 &= \sqrt{2}(y_p - y_q) + x_q - y_p - (y_p - y_q).
\end{aligned}$$

Therefore

$$\begin{aligned}
||pq||_4 &= \sqrt{2}(y_p - y_q) + x_q - x_p - (y_p - y_q) \\
&= (\sqrt{2}-1)y_p + (1-\sqrt{2})y_q + x_q - x_p \\
&< (\sqrt{2}-2)y_p + (\sqrt{2}-1)y_q + x_q \quad (\text{since } x_p > y_q) \\
&\leqslant x_q - y_q + \sqrt{2}y_q = ||sq||_4.
\end{aligned}$$

Therefore, Case 2 is proved. We can also prove other cases (point p is in the region R_1 or R_2 with reference to q) in similar ways.

Since each region R_i has the uniqueness property, we can design a sweep line algorithm to construct the spanning graph. Its pseudocode is presented in Fig. 7 and works as follows. In order to find the closest points in each region for all points, the

```
Algorithm Octilinear Spanning Graph (OSG)
  for (i = 1; i < 5; i++){
    if (i == 1) sort points according to sqrt(2)*x+y-x;
    if (i == 2) sort points according to sqrt(2)*y+x-y;
    if (i == 3) sort points according to sqrt(2)*abs(y)+x+y;
    if (i == 4) sort points according to sqrt(2)*x-y-x;
    A[i] = NULL;
    for each point p in the order{
      find the subset of points in A[i] such that p is in their R[i] region;
      connect p with points in the subset;
      delete the subset from A[i];
      add p to A[i];
    }
  }
```

Fig. 7. Octilinear spanning graph algorithm.

points will be sorted in nondecreasing order of their distance to an imaging point. For example, in region R_1, we set the imaging point at the position $(-\text{MAX}, -\text{MAX})$ (MAX is the maximal value of point coordinates) and sort the points by $\sqrt{2}*x+y-x$. In this order, after each point p is swept, the first point seen in its R_i region is its closest point in that region, which will be connected to p. We use an active set A_i to keep the swept points waiting for closest points. When a new point is swept, we search A_i to find points with the new point in their R_i region. These found points will be deleted from A_i after adding edges from the new point to them. Then, the new point will be added to A_i to wait for its closest point. We only need to consider regions R_1 through R_4 in our algorithm because the connections in other regions have been implied by connections in these regions.

After the spanning graph is constructed, we can get an octilinear minimal spanning tree easily by using some graph-based approaches, such as the Prim algorithm or Kruskal's algorithm.

IV. Octilinear Steiner Tree Construction

We choose Kruskal's algorithm to construct a minimal spanning tree on the spanning graph because it can be used with merging binary tree to find the longest edges in formed cycles [28], which is a crucial part both in OST-E and in the OST-T algorithm.

Kruskal's algorithm first sorts all the edges of a spanning graph by nondecreasing length and then considers them in the order. If the ends of the current edge have not been connected, the edge will be included in the minimal spanning tree; otherwise, it will be excluded. We can represent these connection operations by a binary tree, where the leaves represent the points, and the internal nodes represent the edges. For each internal node, its two children represent the two components connected by the corresponding edge in the minimal spanning tree. To illustrate this, a spanning tree and its merging binary tree are shown in Fig. 8. As we can see, the longest edge between any 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. 8 is (b, c).

In our OST-E (octilinear Steiner tree by edge substitution) algorithm, to find the longest edge in the cycle formed by connecting a point to an edge, we need to find which end of the edge is in the same component with that point before connecting the edge. The least common ancestor of these two points is the longest edge that needs to be deleted. For example, after connecting p with edge (a, b), because p and b are in the same component before connecting (a, b), the longest edge to be deleted in the cycle is (b, c).

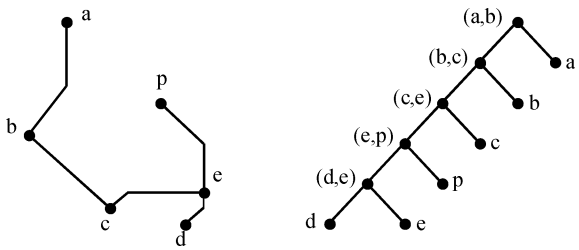


Fig. 8. Minimal spanning tree and its merging binary tree.

It is crucial to find point–edge pairs in the OST-E algorithm and to find triples in the OST-T algorithm. We can utilize the spanning graph to greatly reduce the number of candidates.

Our algorithms are both based on spanning graphs, which can be used not only to construct octilinear minimal spanning trees, but also to get point–edge pairs in OST-E or triples in OST-T.

For edge substitution, in order to reduce the number of point–edge candidates from $O(n^2)$ to $O(n)$, Borah *et al.* [23] suggested using the visibility of a point from an edge, that is, only a point visible from an edge can be connected to that edge. This requires a sweep line algorithm to find visibility relations between points and edges. A crucial observation is that if a point is visible to an edge, then the point is usually connected to at least one end of that edge in the spanning graph [31]. Thus, for each edge in the minimal spanning tree, we may just consider the neighbors of either end of the edge to form point–edge pairs. Since the cardinality of the spanning graph is $O(n)$, the number of possible point–edge pairs generated by this way is also $O(n)$. Then, the longest edge can be removed as shown in Fig. 9.

For triple contraction, Kahng *et al.* [19] proposed a divide-and-conquer algorithm, which can compute a set of $O(n \log n)$ triples containing all empty tree triples in $O(n \log n)$ time [25]. An empty tree triple is the triple that does not contain any other terminals in the minimum rectangle bounding it. As shown in [25], there are at most $36n$ empty tree triples. These triples are sufficient to get a nearly optimal solution for triple contraction. We observed that an empty triple usually has at least two connected edges in the spanning graph. This observation is similar to the observation in OST-E. As shown in Fig. 10, triple (p, a, b)

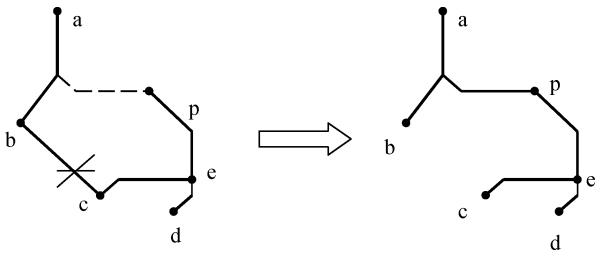


Fig. 9. Diagram showing removal of longest edge.

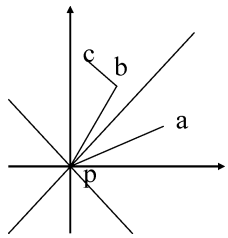


Fig. 10. Empty triple, usually has at least two connected edges in the spanning graph.

and triple (p, b, c) are two examples. Therefore, in our algorithm, we use the edges of the spanning graph to get triples. For each edge in the minimal spanning tree, all points that are neighbors of either end point in a spanning graph will be considered to form triples with this edge. Because the cardinality of a spanning graph is $O(n)$, we will have a set of $O(n)$ triples, which nearly contains all the empty triples. Furthermore, if point–edge pairs are considered as triples, they are a subset of triples considered in the OST-T. Thus, the OST-T theoretically has a better performance than OST-E, but it will be slower since it considers more triples.

In the OST-T (octilinear Steiner tree by triple contraction) algorithm, while we contract a triple, that is, connect three points by a Steiner point, two cycles will be formed in the graph. Thus, we should find the two longest edges, corresponding to two least common ancestors in the merging binary tree. Therefore, we need to choose two two-point pairs. One way is to find the least common ancestors of the lowest point with the other two. For example, in Fig. 8, if the triple of points p, a, b is contracted, we can consider pairs p, a and p, b to get the two longest edges (a, b) and (b, c). The other way is to find the least common ancestor of each two-point pair. Among the three edges we got, two of them should be the same. For example, also in Fig. 8, the three edges corresponding to three two-point pairs are (a, b), (a, b), and (b, c). We can omit the repeat edge and get the two longest edges to delete.

Based on the previous discussion, the pseudocode of these two algorithms is described in Fig. 11. At the beginning of each algorithm, the octilinear spanning graph (OSG) is generated. Then, Kruskal's algorithm is used on the graph to generate a minimal spanning tree and the corresponding merging binary tree. Procedure find_set (u) returns a representative element from the set that contains u; procedure lca_add_edge() records the connection of two points to the merging binary tree; and procedure union_set() does the union operation on two sets. In OST-E, during this process, point–edge pairs will also be added to the query list by procedure lca_add_query(). In addition, by using Tarjan's offline least common ancestor algorithm [27], we can get the longest edge to be deleted for each pair in procedure lca_answer_queries(). Thus, the gain of each point–edge pair can be calculated and sorted. Then, we can consider these sorted point–edge pairs in turn. If neither the connection edge nor the deletion edge has been deleted, a Steiner point can be added, and a connection will be realized.

In OST-T, we add all the triples formed by an edge of the spanning graph and its neighbor points to the query list by procedure lca_add_queries(). The two longest edges in the cycles formed by contracting the triple can be found in procedure lca_answer_queries(); thus, the gain of each triple can be calculated and sorted. Although we use the same procedure name lca_add_queries() and lca_answer_queries() as OST-E, the operations are different. After we get the sorted sequence of triples, we can consider them in turn. If both longest edges have not been deleted, a Steiner point and three new edges will be added.

The running time of these two algorithms are dominated by spanning graph generation and edge sorting, which takes $O(n \log n)$. In addition, 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 [27], which grows extremely slow.

V. Extensions

Because our approaches are graph based, they can be easily extended to many variants of Steiner tree problem. For example, our approaches can be used to handle obstacles by deleting some edges in the spanning graph. This feature is not available in other approaches.

Also as mentioned previously, these approaches can be extended to any λ geometry. We will discuss the spanning graph construction in any λ-geometry plane in following paragraphs.

It is observed that the contour of equidistant points is helpful for the spanning graph construction. For the general λ-geometry case, this equidistant contour will form a polygon that is regular and has 2λ sides. To prove this, we can divide the plane into 2λ regions, as shown in Fig. 12.

Each border between two regions forms a $(\pi/\lambda) * k$-degree angle with positive x axis $(k = 0, 1, 2 \ldots 2\lambda - 1)$. For the sake of simplifying the proof, we will only consider the case for region R_1. Let $Sb = Sa = r$ so that all the points on segment ab are at a distance of r from point S (such as point C, which we can prove by just adding a segment cd, $cd \parallel sb$). Thus, the contour of all the points at the same distance from origin forms a regular polygon.

Furthermore, the division of these regions is a guide for spanning graph construction. We can prove each region Ri has the uniqueness property for $\lambda \geq 3$.

Theorem 2: For any λ geometry $(\lambda \geq 3)$, each region R_i $(i = 1, 2 \ldots 2\lambda)$, shown in Fig. 12, has the uniqueness property. That is, for any points $p, q \in R_i$, $\|pq\|_\lambda < \max(\|sp\|_\lambda, \|sq\|_\lambda)$.

Proof: In the λ-geometry plane, the connections between points must follow the legal directions, which have an angle of $(\pi/\lambda) * k$ $(k = 0, 1, 2 \ldots 2\lambda - 1)$ with the positive x axis. A

```
Algorithm Octilinear Steiner Tree with Edge Substitution (OST-E)
    T = NULL;
    Generate the spanning graph G by OSG algorithm;
    For (each edge (u,v)of 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)
                lca_add_query(w, (u,v));
            lca_add_edge((u,v), s1.edge); lca_add_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) have not been deleted from T){
            connect p to a, b by adding three edges and one Steiner point to T;
            delete (a,b), (c,d) from T;
        }
    Output T;

Algorithm Octilinear Steiner Tree with Triple Contraction (OST-T)
    T = NULL;
    Generate the spanning graph G by OSG algorithm;
    Generate the minimal spanning tree T and merging binary tree by Kruskal's algorithm,
    For (each edge (u,v)of G in non-decreasing length)
        for (each neighbor w of u,v in G)
            lca_add_query(w, u, v);
    Generate triple info by lca_answer_queries;
    For (each triple info (w, u, v, (a,b), (c,d))) in non-increasing positive gains)
        if ((a,b), (c,d) have not been deleted from T){
            connect w, u, v by adding three edges and one Steiner point to T;
            delete (a,b), (c,d) from T;
        }
    Output T;
```

Fig. 11. OST-E and OST-T algorithms.

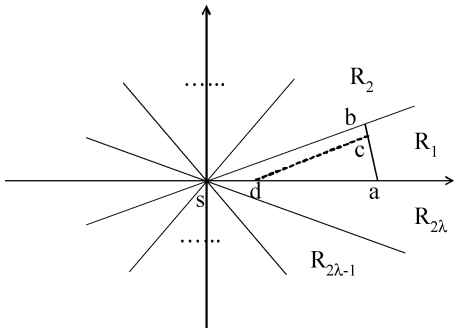


Fig. 12. Equidistant points in λ geometry.

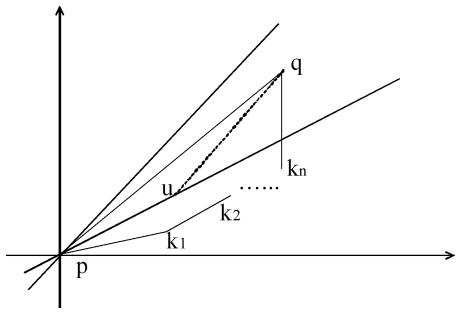


Fig. 13. Lemma 1.

line segment following legal direction is a legal line segment. A path is legal if all the line segments on it are legal. We call two directions (or line segments) adjacent if they have have the angles of $(\pi/\lambda) * i$ and $(\pi/\lambda) * j$ with the positive x axis such that $((i-j) \bmod 2\lambda) = 1$ or $2\lambda - 1$.We first give two lemmas as follows:

Lemma 1: The distance between two points p, q in the λ-geometry plane (denoted by $||pq||_\lambda$) either equals to their distance in Euclidean plane (denoted by $||pq||_\infty$) or equals to the sum of two adjacent legal line segments in the λ-geometry plane.

Proof: If pq follows the legal direction, then $||pq||_\lambda = ||pq||_\infty$ by definition. If not, we can assume pq is between two adjacent legal line segments, as shown in Fig. 13. It was proved in [29] that $||pq||_\lambda = ||pu||_\lambda + ||uq||_\lambda$; here, pu and uq follow two adjacent legal directions and make a minimal possible angle with pq. This legal path is shortest when compared with other legal paths from p to q, such as $p -> k_1 -> k_2 -> \ldots -> k_n -> q$. □

Lemma 2: For point pairs (p, q) and (p', q'), if the connection of p, q is parallel to the connection of p', q' in Euclidean plane, i.e., $pq \parallel p'q'$, then we get $||pq||_\infty / ||p'q'||_\infty = ||pq||_\lambda / ||p'q'||_\lambda$.

Proof: We can set p' at the same place as p. If pq follows the legal direction, then $||pq||_\lambda = ||pq||_\infty$ and $||p'q'||_\lambda = ||p'q'||_\infty$, so $||pq||_\infty / ||p'q'||_\infty = ||pq||_\lambda / ||p'q'||_\lambda$. If not, as shown in Fig. 14, there exist pu and uq, which satisfy $||pq||_\lambda = ||pu||_\lambda + ||uq||_\lambda$; also there exist $p'u'$ and $u'q'$, which

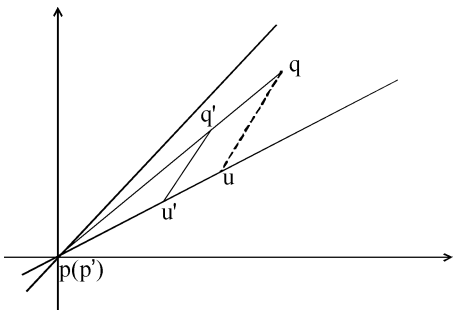


Fig. 14. Lemma 2.

satisfy $||p'q'||_\lambda = ||p'u'||_\lambda + ||u'q'||_\lambda$. Since triangle pqu is similar to triangle $p'q'u'$, i.e.,

$$\frac{||pq||_\infty}{||p'q'||_\infty} = \frac{||pu||_\infty}{||p'u'||_\infty} = \frac{||uq||_\infty}{||u'q'||_\infty}$$

we can get

$$\begin{aligned}\frac{||pq||_\infty}{||p'q'||_\infty} &= \frac{(||pu||_\infty + ||uq||_\infty)}{(||p'u'||_\infty + ||u'q'||_\infty)} \\ &= \frac{(||pu||_\lambda + ||uq||_\lambda)}{(||p'u'||_\lambda + ||u'q'||_\lambda)} \\ &\quad (\text{since } pu, uq, p'u', u'q' \text{ all follow} \\ &\quad \text{legal directions}) \\ &= \frac{||pq||_\lambda}{||p'q'||_\lambda}\end{aligned}$$

□

Then, we start to prove our theorem. Since regions R_1 through $R_{2\lambda}$ are equivalent to each other, we only give a proof to R_1 for convenience. In Fig. 15, $p, q \in R_1$. We assume $y_p \geq y_q$; then, there are three cases shown in Fig. 15(a)–(c), depending on the angle between pq and the positive x axis, which is denoted by α. In addition, we assume that region R_1 does not include the bounding half line between R_1 and R_2.

If $\alpha \leq \pi/\lambda$, as shown in Fig. 15(a), then

$$||pq||_\lambda = ||pv||_\lambda + ||qv||_\lambda < ||pu||_\lambda + ||su||_\lambda = ||sp||_\lambda.$$

Therefore, we get $||pq||_\lambda < \max(||sp||_\lambda, ||sq||_\lambda)$ in this case.

If $\alpha > \pi/\lambda$, we can transform the original problem to a new problem, in which p, q are on the line with legal direction, as shown in Fig. 15(b) and (c).

In Fig. 15(b), $\pi/\lambda < \alpha \leq \pi/2$. Let $pq \parallel p'q' \parallel p''q''$; then, we can get

$$\begin{aligned}&||pq||_\lambda \leq ||p'q'||_\lambda = ||p''q''||_\lambda \quad (\text{by Lemma 2}) \\ &||sp||_\lambda = ||pp''||_\lambda + ||sp''||_\lambda > ||sp''||_\lambda \quad (\text{by Lemma 1}) \\ &||sq||_\lambda = ||sq'||_\lambda + ||qq'||_\lambda > ||sq''||_\lambda \quad (\text{by Lemma 1}).\end{aligned}$$

In Fig. 15(c), $\pi/2 < \alpha < \pi$. Let $pq \parallel p'q' \parallel p''q''$; then we can get

$$\begin{aligned}&||pq||_\lambda \leq ||p'q'||_\lambda = ||p''q''||_\lambda \quad (\text{by Lemma 2}) \\ &||sp||_\lambda = ||pp'||_\lambda + ||sp'||_\lambda > ||sp''||_\lambda \quad (\text{by Lemma 1}) \\ &||sq||_\lambda = ||sq''||_\lambda + ||qq''||_\lambda > ||sq''||_\lambda \quad (\text{by Lemma 1}).\end{aligned}$$

Therefore, for the cases in Fig. 15(b) and (c), if we can prove $||p''q''||_\lambda \leqslant \max(||sp''||_\lambda, ||sq''||_\lambda)$, and then we can prove the original theorem $||pq||_\lambda < \max(||sp||_\lambda, ||sq||_\lambda)$. Here, p'', q'' are on the direction (π/λ) and positive x axis, respectively.

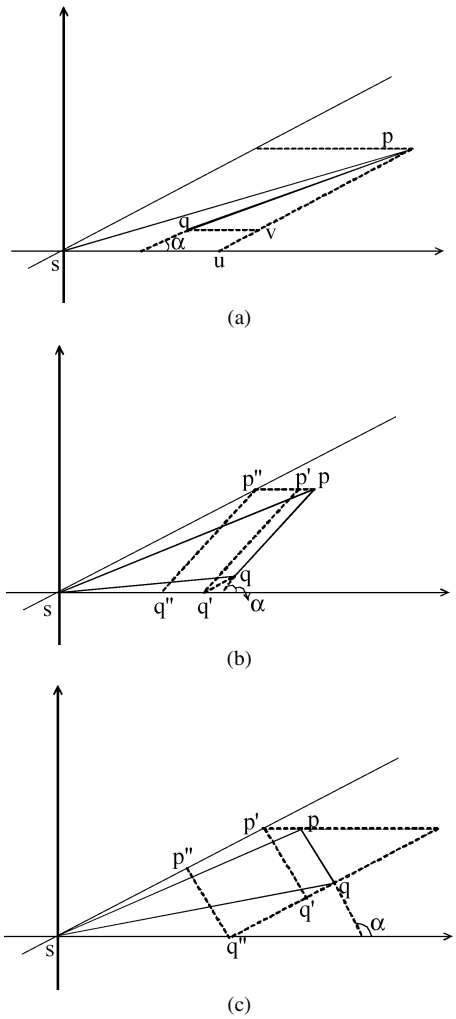


Fig. 15. (a) $\alpha \leq \pi/\lambda$. (b) $\pi/\lambda < \alpha \leq \pi/2$. $\pi/2 < \alpha < \pi$.

For this $||p''q''||_\lambda \leqslant \max(||sp''||_\lambda, ||sq''||_\lambda)$ problem, we also denote the angle between $p''q''$ and the positive x axis as angle α, which is larger than π/λ [the case $\alpha \leq \pi/\lambda$ is solved in Fig. 15(a)]. Then, we consider two cases as shown in Fig. 16.

In Fig. 16(a), $\pi/\lambda < \alpha \leq (\pi - \pi/\lambda)$. Let $su \parallel p''q''$, and $up'' \parallel sq''$. sm and sn are two adjacent legal directions. w is on sp'', which is the legal direction with angle π/λ to a positive x axis. Thus

$$\begin{aligned}||p''q''||_\lambda &= ||us||_\lambda \\ &= ||uv||_\lambda + ||vs||_\lambda \\ &\leq ||uv||_\lambda + ||vw||_\lambda + ||ws||_\lambda \\ &= ||uw||_\lambda + ||ws||_\lambda.\end{aligned}$$

sm is the legal direction on or next to line su, which forms angle α with the x axis, and $\alpha \leq \pi - \pi/\lambda$. Thus, $\angle msq'' \leq \pi - \pi/\lambda$ (note that if $\alpha = \pi - \pi/\lambda$, then $\angle msq'' = \pi - \pi/\lambda$).

Since $\angle\beta = \pi/\lambda$ and $\angle\gamma = \pi - \angle msq'' \geq \pi/\lambda$, then

$$||p''q''||_\lambda \leq ||uw||_\lambda + ||ws||_\lambda \leq ||p''w||_\lambda + ||ws||_\lambda = ||sp''||_\lambda.$$

Therefore, we prove $||p''q''||_\lambda \leqslant \max(||sp''||_\lambda, ||sq''||_\lambda)$ for this case.

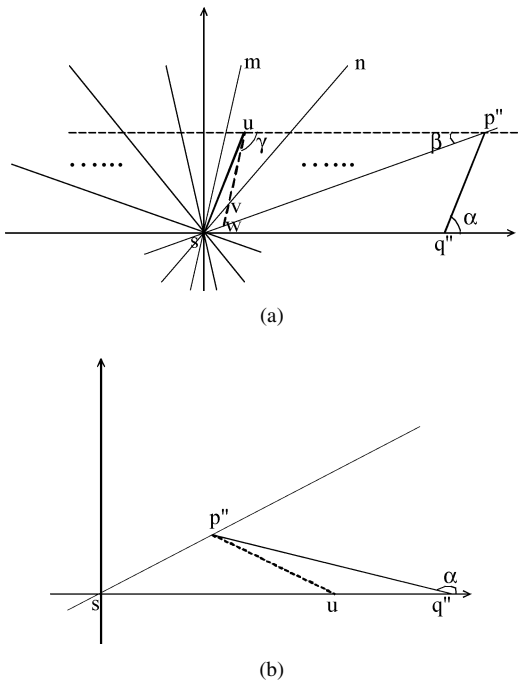


Fig. 16. (a) $\pi/\lambda < \alpha \leq \pi - \pi/\lambda$. (b) $(\pi - \pi/\lambda) < \alpha < \pi$.

In Fig. 16(b), $(\pi - \pi/\lambda) < \alpha < \pi$. Therefore, $||p''q''||_\lambda = ||p''u||_\lambda + ||q''u||_\lambda$; here, $p''u$ and $q''u$ are two adjacent legal directions.

Furthermore, since $\lambda \geq 3$, angle $sp''u = \pi - 2\pi/\lambda \geq \pi/\lambda$, then

$$||p''q''||_\lambda = ||p''u||_\lambda + ||q''u||_\lambda \leq ||su||_\lambda + ||uq''||_\lambda = ||sq''||_\lambda.$$

Therefore, we prove $||p''q''||_\lambda \leqslant \max(||sp''||_\lambda, ||sq''||_\lambda)$ for this case.

We proved $||p''q''||_\lambda \leqslant \max(||sp''||_\lambda, ||sq''||_\lambda)$ for the above two cases, so we also proved original Theorem 2, i.e., uniqueness property: $||pq||_\lambda < \max(||sp||_\lambda, ||sq||_\lambda)$.

The method to construct a spanning graph in a general λ-geometry plane is as follows.

First, based on the polygon that is formed by equidistant points, we can divide the plane into several sections. Because each region has the uniqueness property, we can sort the points in each region and scan them to construct the spanning graph. On the other hand, we can only scan the points within one region, because the points in different region have different expression of distance from the origin.

For the running time and storage in general λ geometry (λ is finite), it runs in $O(\lambda n \log n)$ time and requires just $O(\lambda n)$ storage.

After we have the spanning graphs, we can use our Steiner tree construction algorithms similarly in any λ geometry.

VI. Experimental Results

We implemented the OSG algorithm and two octilinear steiner tree algorithms (OST-E and OST-T) in C language, following the pseudocode in Figs. 7 and 11, with the exception that the program starting from the first "for" loop in Fig. 11 will be run repeatedly until no improvement.

TABLE I
Percent Improvement Over Rectilinear Minimal Spanning Tree (RMST)

Input Size	RMST	OST-E	OST-T
	Length	Improve (%)	Improve (%)
Random Instances (average results over 10 experiments)			
10	24215	20.16932	20.16932
100	81679	18.10012	18.16256
200	117787	18.88663	18.89088
300	143930	18.49163	18.49649
1000	259277	19.42826	19.43751
2000	365003	19.21162	19.23107
3000	443622	19.0739	19.10185
5000	570360	19.03061	19.05796
10000	81190917	19.01136	19.01971
50000	181146384	19.14284	19.16234
100000	255478245	19.13443	19.15327
VLSI Instances (average results over 10 experiments)			
337	24773	14.23727	14.23727
1944	45225	12.86014	12.9508
2437	57879	12.68681	12.88377
2676	88723	14.97582	15.00513
12052	265274	13.79404	13.83588
22373	1.396E+09	16.80561	16.82671

In Table I, we compare our octilinear Steiner tree results with RMST. In Table II, we report our comparisons between our two programs with Kahng's octilinear Steiner tree program—batched greedy, which has a $O(n \log^2 n)$ running time and good performance. For fair comparisons, we compiled and ran all the programs on the same machine—a Dell Precision 650 with 4 Gb of memory and two 3.06-GHz CPUs. The test bed for our experiments consists of two categories: random test cases ranging from 10 to 100 000 terminals and test cases extracted from recent industrial designs, ranging in sizes from 330 to 23 000. For each input size, we report the average improvement ratio of the Steiner tree over minimal spanning tree (in percentage) and average running time (in seconds) of ten repeated experiments.

The results in Table I show that our two octilinear approaches can greatly reduce the wire length for test cases with different size. The average reduction rate over RMST is around 19%, higher than most previous algorithms, such as the GRATS-tree algorithm [15].

Table II shows that both OST-E and OST-T are faster than Kahng's batched greedy. For the test case with 22 373 terminals, OST-E is 5.3 times faster, and OST-T is 1.7 times faster than batched greedy. The table also shows that OST-E has a similar performance as batched greedy, and OST-T has a better performance than batched greedy in some cases.

VII. Conclusion

In summary, we developed two efficient octilinear Steiner tree algorithms that are based on Zhou *et al.*'s spanning graph and combined with Borah *et al.*'s edge substitution and Zelikovsky's triple contraction, respectively. The implementations have a running time of $O(n \log n)$ and a storage requirement of $O(n)$, without a large hidden constant.

TABLE II
PERCENT IMPROVEMENT OVER OCTILINEAR MINIMAL SPANNING TREE (OMST) AND CPU TIME OF EACH PROGRAM

Input	OMST	OST-E		Batched Greedy		OST-T	
	Length	Imp (%)	CPU (sec.)	Imp (%)	CPU (sec.)	Imp (%)	CPU (sec.)
Random Instances (average results over 10 experiments)							
10	20040	3.537924	< 0.01	3.537924	< 0.01	3.537924	< 0.01
100	70167	4.663161	<0.01	4.663161	<0.01	4.735844	<0.01
200	99935	4.396858	<0.01	4.453895	0.02	4.401861	0.01
300	122122	3.936228	<0.01	3.994366	0.05	3.941960	0.01
1000	218222	4.269964	0.06	4.298375	0.23	4.280962	0.09
2000	308286	4.348559	0.22	4.358615	1.82	4.371590	1.42
3000	375336	4.350768	0.35	4.368086	1.10	4.383805	0.65
5000	482424	4.271554	0.62	4.315084	1.57	4.303890	1.15
10000	68720982	4.315372	1.71	4.335769	5.10	4.325231	3.59
50000	153045172	4.296345	11.78	*	*	4.319435	34.18
100000	215853782	4.289864	20.71	*	*	4.312172	63.70
VLSI Instances (average results over 10 experiments)							
337	21900	2.986301	<0.01	2.986301	0.04	2.986301	0.01
1944	40723	3.226678	0.18	3.469784	0.52	3.327358	0.27
2437	52311	3.393168	0.24	3.777408	0.58	3.611095	0.35
2676	78019	3.310732	0.28	3.506838	0.92	3.344057	0.40
12052	237228	3.602442	1.41	3.713727	4.52	3.649232	2.05
22373	1.207E+09	3.759414	2.94	3.741423	15.17	3.778159	8.67

(Asterisk * indicates that the results cannot be calculated in the given time, i.e., 1000 seconds)

Our future work will focus on the Steiner tree construction avoiding obstacles.

REFERENCES

[1] R. C. Carden IV, J. M. Li, and C. K. Cheng, "A global router with a theoretical bound on the optimal solution," *IEEE Trans. Computer-Aided Design Integr. Circuits Syst.*, vol. 15, no. 2, pp. 208–216, Feb. 1996.

[2] T. Jing, X. L. Hong, H. Y. Bao, J. Y. Xu, and J. Gu, "SSTT: efficient local search for GSI global routing," *J. Comput. Sci. Technol.*, vol. 18, no. 5, pp. 632–639, 2003.

[3] T. Jing, X. L. Hong, J. Y. Xu, H. Y. Bao, C. K. Cheng, and J. Gu, "UTACO: A unified timing and congestion optimization algorithm for standard cell global routing," *IEEE Trans. Computer-Aided Design Integr. Circuits Syst.*, vol. 23, no. 3, pp. 358–365, Mar. 2004.

[4] S. P. Lin and Y. W. Chang, "A novel framework for multilevel routing considering routability and performance," in *Proc. IEEE/ACM Int. Conf. Computer Aided Design (ICCAD)*, 2002, pp. 44–50.

[5] C. C. N. Chu and D. F. Wong, "An efficient and optimal algorithm for simultaneous buffer and wire sizing," *IEEE Trans. Computer-Aided Design Integr. Circuits Syst.*, vol. 18, no. 9, pp. 1297–1304, Sep. 1999.

[6] H. Xiang, X. P. Tang, and D. F. Wong, "An algorithm for integrated pin assignment and buffer planning," in *Proc. ACM/IEEE Design Automation Conf. (DAC)*, 2002, pp. 584–589.

[7] X. L. Hong, T. Jing, J. Y. Xu, H. Y. Bao, and J. Gu, "CNB: A critical-network-based timing optimization method for standard cell global routing," *J. Comput. Sci. Technol.*, vol. 18, no. 6, pp. 732–738, 2003.

[8] J. Y. Xu, X. L. Hong, T. Jing, Y. C. Cai, and J. Gu, "A novel timing-driven global routing algorithm considering coupling effects for high performance circuit design," *IEICE Trans. Fundamentals ECCS*, vol. E86-A, no. 12, pp. 3158–3167, 2003.

[9] Y. Wang, X. L. Hong, T. Jing, Y. Yang, X. D. Hu, and G. Y. Yan, "An efficient low-degree RMST algorithm for VLSI/ULSI physical design," in Lecture Notes in Computer Science (LNCS) 3254—Integrated Circuit and System Design, Springer, Germany, Santorini, Greece, Sep. 2004, pp. 442–452.

[10] J. Y. Xu, X. L. Hong, T. Jing, Y. C. Cai, and J. Gu, "An efficient hierarchical timing-driven Steiner tree algorithm for global routing," *Integration, VLSI J.*, vol. 35, no. 2, pp. 69–84, 2003.

[11] M. Sarrafzadeh and C. K. Wong, "Hierarchical Steiner tree construction in uniform orientations," *IEEE Trans. Computer-Aided Design Integr. Circuits Syst*, vol. 11, no. 9, pp. 1095–1103, Sep. 1992.

[12] C. K. Koh, "Steiner problem in octilinear routing model," Master's thesis, Dept. of Computer Science, National Univ. of Singapore, Singapore, 1995.

[13] D. T. Lee, C. F. Shen, and C. L. Ding, "On Steiner tree problem with 45° routing," in *Proc IEEE Int. Symp. Circuits Systems (ISCAS)*, Seattle, WA, 1995, pp. 1680–1682.

[14] G. Robins, "On optimal interconnections," Ph.D. dissertation, Dept. of Computer Science, Univ. of California, Los Angeles, 1992.

[15] C. K. Koh and P. H. Madden, "Manhattan or non-Manhattan? A study of alternative VLSI routing architectures," in *Proc. Great Lakes Symp. VLSI*, San Diego, CA, 2000, pp. 47–52.

[16] S. L. Teig, "The X architecture: Not your father's diagonal wiring," in *Int. Workshop System Level Interconnect Prediction (SLIP)*, San Diego, CA, 2002, pp. 33–37.

[17] C. Chiang and C. S. Chiang, "Octilinear Steiner tree construction," presented at the IEEE Int. Midwest Symp. Circuits Systems (MWSCAS), Tulsa, OK, 2002, Paper TAM1l-216.

[18] . [Online]. Available: http://www.xinitiative.org/

[19] A. B. Kahng, I. I. Mandoiu, and A. Z. Zelikovsky, "Highly scalable algorithms for rectilinear and octilinear Steiner trees," presented at the Asia–Pacific Design Automation Conf. (ASP-DAC), Kitakyushu, Japan, Jan. 2003.

[20] A. Zelikovsky, "An 11/6-approximation algorithm for the network Steiner problem," *Algorithmica*, vol. 9, pp. 463–470, 1993.

[21] A. B. Kahng and G. Robins, "A new class of Steiner tree heuristics with good performance," *IEEE Trans. Computer-Aided Design Integr. Circuits Syst.*, vol. 11, no. 7, pp. 893–902, Jul. 1992.

[22] C. S. Coulston, "Constructing exact octagonal Steiner minimal tree," presented at the Great Lakes Symp. VLSI (GLSVLSI), Washington, DC, Apr. 28–29, 2003.

[23] M. Borah, R. M. Owens, and M. J. Irwin, "An edge-based heuristic for Steiner routing," *IEEE Trans. Computer-Aided Design Integr. Circuits Syst.*, vol. 13, no. 12, pp. 1563–1568, Dec. 1994.

[24] H. Zhou, N. Shenoy, and W. Nicholls, "Efficient spanning tree construction without Delaney triangulation," *Inf. Process. Lett.*, vol. 81, no. 5, 2002.

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

[26] U. Foßmeier, M. Kaufmann, and A. Zelikovsky, "Faster approximation algorithms for the rectilinear Steiner tree problem," *Discrete Computational Geometry*, vol. 18, no. 1, pp. 93–109, 1997.

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

[28] H. Zhou, "Efficient Steiner tree construction based on spanning graphs," presented at the ACM Int. Symp. Physical Design, Monterey, CA, Apr. 2003.

[29] P. Widmayer, Y. F. Wu, and C. K. Wong, "On some distance problems in fixed orientations," *SIAM J. Comp.*, vol. 16, no. 4, pp. 728–746, Aug. 1987.

[30] Q. Zhu, H. Zhou, T. Jing, X. Hong, and Y. Yang, "Efficient octilinear Steiner tree construction based on spanning graphs," in *Proc. IEEE/ACM Asian–Pacific Design Automation Conf. (ASP-DAC)*, Yokohama, Japan, 2004, pp. 687–690.

[31] H. Zhou, "Efficient Steiner tree construction based on spanning graphs," *IEEE Trans. Computer-Aided Design Integr. Circuits Syst.*, vol. 23, no. 5, pp. 704–710, May 2004.

Qi Zhu (S'03) received the B.E. degree in computer science and technology from Tsinghua University, Beijing, China, in 2003. He is currently working toward the Ph.D. degree in the Department of Electrical Engineering and Computer Sciences, University of California, Berkeley.

His research interest focuses on embedded system design, physical synthesis, and system-level design.

Hai Zhou (SM'04) 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 at Austin in 1999.

Previously, he was with the Advanced Technology Group at Synopsys, Inc., Mountain View, CA. He is currently an Assistant Professor in electrical and computer engineering at Northwestern University, Evanston, IL. His research interests include very large scale integration (VLSI) computer-aided design, algorithm design, and formal methods.

Dr. Zhou was a recipient of a CAREER Award from the National Science Foundation in 2003. He has served on the technical program committees of the Association for Computing Machinery (ACM) International Symposium on Physical Design and the IEEE International Conference on Computer-Aided Design.

Tong Jing (M'02) received the B.S. degree in electronic engineering (EE) and the M.S. and Ph.D. degrees in computer science (CS) from Northwestern Polytechnical University (NPU), Xi'an, China, in 1989, 1992, and 1999, respectively.

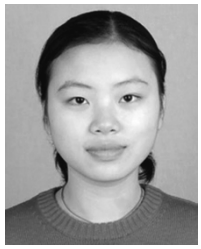
From September 1999 to April 2001, he was a Postdoctoral Researcher of the Electronic Design Automation (EDA) Laboratory in the Computer Science and Technology (CST) Department, Tsinghua University, Beijing, China. Since 2001, he has been a Faculty Member in the CST Department at Tsinghua University, where he is currently an Associate Professor. From December 2000 to February 2001, he was a Visiting Scholar in the Computer Science and Engineering (CSE) Department, University of California, San Diego (UCSD). He has authored or coauthored more than 80 papers published in technical journals and conference proceedings. His research interests include performance optimization routing, routability, Steiner problem, interconnect optimization, and signal integrity.

Dr. Jing is a recipient of the IEEE International Conference on ASIC (ASICON) Outstanding Student Paper Award in 2003. He is a recipient of the First Class Award for Excellence in Teaching from Tsinghua University in 2002 and 2004. He has served as the Secretary General and Chair of Physical Design and Interconnect Optimization Technical Program Committee (TPC) Subcommittee of the 2005 IEEE/Association for Computer Machinery (ACM) Asian–Pacific Design Automation Conference (ASP-DAC), a TPC member, a Panel Speaker, and Session Co-Chair of the IEEE International Conference on Communications, Circuits and Systems (ICCCAS) 2004; a Session Chair of the International Symposium Computing and Information (ISCI) 2004; the Secretary General and TPC member of the 2003 IEEE ASICON; and the Session Co-Chair of the 2003 IEEE/ACM ASP-DAC.

Xian-Long Hong (M'95–SM'95–F'03) received the B.S. degree from Tsinghua University, Beijing, China, in 1964.

Since 1988, he has been a Professor in the Department of Computer Science and Technology, Tsinghua University, Beijing, China. His research interests include very large scale integration (VLSI) layout algorithms and design automation systems.

Dr. Hong is a Senior Member of the Chinese Institute of Electronics. He has been an Associate Editor of IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—PART I since 2002.



Yang Yang received the B.S. degree in computer science and technology from Tsinghua University, Beijing, China, in 2003. She is currently working toward the M.S. degree at the Design Automation Laboratory, Computer Science and Technology Department, Tsinghua University.

Her research interests focus on computer-aided design of very large scale integration (VLSI) circuits and systems and computer architecture.