# The Polygonal Contraction Heuristic for Rectilinear Steiner Tree Construction*

Yin Wang, Xianlong Hong, Tong Jing, Yang Yang

Department of Computer Science and Technology
Tsinghua University
Beijing 100084, P. R. China
Email: {wang-y01, yycs99}@mails.tsinghua.edu.cn
{hxl-dcs, jingtong}@tsinghua.edu.cn

Xiaodong Hu, Guiying Yan

Institution of Applied Mathematics
Chinese Academy of Sciences
Beijing 100080, P. R. China
Email: xdhu@public.bta.net.cn
yangy@mail.amss.ac.cn

**Abstract— Motivated by VLSI/ULSI routing applications, we present a heuristic for rectilinear Steiner minimal tree (RSMT) construction. We transform a rectilinear minimum spanning tree (RMST) into an RSMT by a novel method called polygonal contraction. Experimental results show that the heuristic matches or exceeds the solution quality of previously best known algorithms and runs much faster.**

## I. INTRODUCTION

Given a set of points in the plane, a Steiner minimal tree (SMT) connects these points through some extra points (the Steiner points) to achieve a minimal total length. The rectilinear Steiner minimal tree (RSMT) problem is a fundamental problem in very/ultra large scale integrated circuit (VLSI/ULSI) design automation, which plays an important role in physical design. A RSMT is generally used in initial net topology creation for global routers, or incremental net tree topology creation in physical synthesis. Useful algorithms have been proposed focusing on SMT [1]–[3], timing-driven RSMT [4], [5], and Obstacle-Avoiding RSMT [6], [7].

As we know, the RSMT problem is proved to be NP-Hard [8]. Although there exists some good exact algorithms such as GeoSteiner [9], in many applications we must compute the RSMT for hundreds of nets and for thousands of times. Moreover, some nets can have thousands of terminals. At these occasions, even the fastest exact algorithm is impractical, and a highly efficient heuristic is preferred. In fact, the SMT problem admits a polynomial-time approximation scheme for any geometric space of constant dimension [10], that is, an approximate solution whose length is at most $(1+\epsilon)$ times that of optimal SMT can be produced in polynomial time of n and $1/\epsilon$. Thus the better the approximation, the larger may be the running time.

Generally, there are two criteria for a heuristic: the speed and the solution quality. For large point sets, the exact RSMT is hard to compute, thus the solution quality of a heuristic is usually measured by the ratio of the sub-optimal Steiner tree length and the MST length. This ratio is usually called the *performance* of the heuristic. There exist several good heuristics. The Batched Iterated 1-Steiner heuristic (BI1S) [11] has a time complexity of $O(n^3)$ and a performance of about 10.9%. The edge-based heuristic of *Borah et al.* [12] has a time complexity of $O(n^2)$ [1] and similar performance as the BI1S. The BGA algorithm by *Kahng et al.* [13] has a time complexity of $O(n \log^2 n)$ and a performance of about 11%. The spanning graph based heuristic of Zhou [14] has a time complexity of $O(n \log n)$ and a performance of about 10.5%.

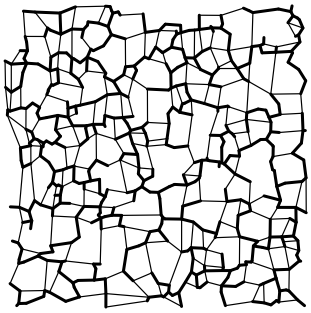


Fig. 1. A BNG on 500 random points and the RMST on it.

In this paper we propose an integrated RSMT heuristic based on the novel idea of *polygonal contraction*. It works as follows. At the beginning, a graph called the bounded-degree neighborhood graph (BNG)is constructed.[2] Then, from the BNG an RMST is built. At last, the RMST is transformed into an RSMT by means of polygonal contraction. Experimental results show that our heuristic is much faster than the previous ones and has a similar or better performance.

The remainder of the paper is organized as follows. Section II gives a brief introduction to the BNG. Then Section III presents an $O(n \log n)$ algorithm to construct the BNG, which leads to an effecient algorithm for RMST construction. And Section IV describes the polygonal con-

*This work was supported in part by the NSFC under Grant No.60373012, the SRFDP of China under Grant No.20020003008, and the Hi-Tech Research and Development (863) Program of China under Grant No.2004AA1Z1050.

[1] Borah also proposed a sweepline scheme to improve the time complexity to $O(n \log n)$. But this requires advanced data structures and has never been implemented.

[2] See Fig. 1 for a BNG and an RMST. Note we draw the RMST edges with straight lines only in favor of easy visualization. We will define the BNG in Section II.

traction method, which transforms this RMST into an RSMT. Finally, Section VI gives some experimental results and Section VII concludes the paper.

## II. The Bounded-degree Neighborhood Graph (BNG)

In our algorithm, we first construct an RMST on the point set and tranform it into an RSMT. In order to construct the RMST efficiently, we first construct a graph on the point set and then find a RMST in this graph. This idea has already been used by many existing metric MST algorithms [15]–[17]. In our algorithm, we propose a graph named the bounded-degree neighborhood graph(BNG), which is a subgraph of the Delaunay triangulation and has the desired property that may aid our RSMT construction.

### A. Definition of the BNG

Before defining the BNG, we define the *weak uniqueness property* as follows.

*Definition 1 (weak uniqueness property):* Given a point $p$, a region $R$ has the *weak uniqueness property* with respect to $p$ if for every pair of points $u$, $w \in R$, we have $\|wu\| \leq \max(\|wp\|, \|up\|)$. A partition of space into a finite set of disjoint regions is said to have the weak uniqueness property if each of its regions has the weak uniqueness property.

We represent a kind of *weak diagonal partition* in Fig. 2(a). The plane is divided into 4 regions by the diagonal lines, and each region has only one diagonal ray in it.

*Theorem 1:* Given a point $p$ in the rectilinear plane, each region of the weak diagonal partition has the weak uniqueness property.

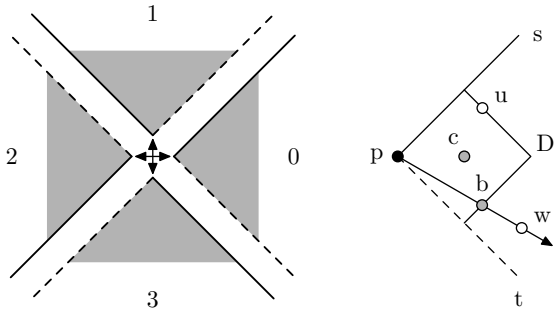
*Proof:* We need to show that for any two distinct points $u$ and $w$ that lie in the same region, we have $\|wu\| \leq \max(\|wp\|, \|up\|)$. Consider $u$, $w$ in one of the regions of weak diagonal partition in Fig. 2(b). Assume without loss of generality that $\|up\| \leq \|wp\|$. Consider the diamond $D$ with left corner at $p$ and center at $c$, such that $u$ is on the boundary of $D$. Let a ray starting at $p$ and passing through $w$ intersect $D$ at $b$. By the triangle inequality, there is $\|wu\| \leq \|wb\| + \|bu\| \leq \|wb\| + \|bc\| + \|cu\| = \|wb\| + \|bc\| + \|cp\| = \|wp\|$. ■

Now the BNG can be defined as follows.

*Definition 2:* The *Bounded-degree Neighborhood Graph (BNG)* is a connected subgraph of the Delaunay triangulation such that if we do a weak diagonal partition at each of its vertices, the center is connected to only the nearest neighbor in each region.

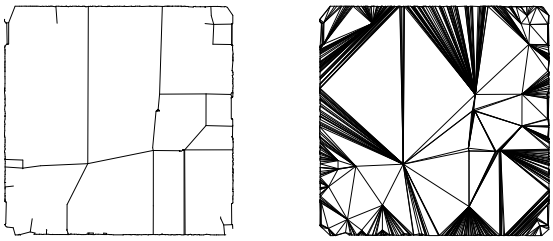
### B. Properties of the BNG

Some properties of the BNG can be simply derived. Since we can have only one edge in each region of a point, obviously the vertex degree of the BNG is bounded by 4. And we are guaranteed to find a 4-BDMST on it if the BNG contains at least one MST.




(a) The weak diagonal partition. (b) The weak uniqueness in a region.

Fig. 2. The weak diagonal partition.



(a) The BNG (b) The Delaunay triangulation

Fig. 3. An bounded-degree neighborhood graph and the corresponding Delaunay triangulation on a point set of size 830 extracted from a real circuit.

Because the maximum vertex degree is 4 and every edge introduce 2 degrees to the graph, the BNG has at most $2n$ edges, where $n$ is the number of vertices in the BNG. On uniformly distributed random point sets, the number of edges is normally below $1.5n$, while the Delaunay triangulation invariably has about $3n$ edges. On a point set extracted from a real circuit design, the number of edges is even smaller. Because the MST algorithms on graphs usually run in time $O(m \log n)$, where $m$ is number of edges, this property leads to a significant speedup of the MST construction. Fig. 3 compares a BNG and a Delaunay triangulation on the same point set.

Because the Delaunay triangulation is planar and the BNG is a subgraph of it, it follows that the BNG is planar, and an $O(n)$ minimum spanning tree algorithm [18] can be applied on it to achieve great efficiency.

## III. Construction of The BNG

We construct the BNG by *pruning* a Delaunay triangulation, which can be constructed in $O(n \log n)$ time with an existing algorithm [19], [20]. At each point $p$, we do a weak diagonal partition. In each region, if there are more than one edge adjacent to $p$, we delete the longer edges, leaving only the shortest one. When there are two edges of equal length, we delete the edge with a larger Euclidean length. When deleting edges, we also merge the triangles of the Delaunay triangulation into polygonal faces, and merge small polygons into bigger polygons. At last, some

Delaunay edges remain, and the resulting subgraph is the BNG. We call this process a *uniqueness pruning process*.

Now we give a formal description of the uniqueness pruning process. The input is a rectilinear Delaunay triangulation, represented in the usual vertex-edge-face data structure[3], and the output is a BNG. We have a two dimensional pointer array PART of size $4n$, where $n$ is the number of points. At anytime in the pruning process, we record in PART[p][r] the shortest edge in region $r$ of the point $p$. Each element of the array is initially a null pointer. We examine the Delaunay edges one by one, testing both end points of the edge. Centered at an end point $p$, we do a weak diagonal partition. If there is already an edge recorded in PART[p][r], we compare its rectilinear length with the edge we are testing; we delete the longer and record the shorter in PART[p][r]. In case of ties we delete the edge with a larger Euclidean length. The BNG construction algorithm can be specified formally as follows.

**Algorithm 1** BNG Construction

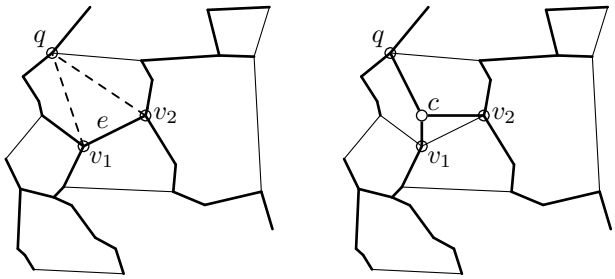
Setup a two dimensional array PART of size $4n$. Construction a Delaunay triangulation.
**for** all each edge $e$ of the Delaunay triangulation **do**
  **for** both of the end point $s$ of $e$ **do**
    **if** e is already deleted **then**
      Return
    **end if**
    r $\leftarrow$ the region number of $s$ in which $e$ lies
    **if** PART[s][r]=NULL **then**
      PART[s][r] $\leftarrow$ e
    **else if** WEIGHT(e) $>$ WEIGHT(PART[s][r]) **then**
      Delete($e$)
      Merge the faces adjacent to $e$
    **else if** WEIGHT(e) $<$ WEIGHT(PART[s][r]) **then**
      Delete(PART[s][r])
      Merge the faces adjacent to the edge PART[s][r]
      PART[s][p] $\leftarrow$ e
    **else**
      Between the choice of $e$ and PART[s][r], delete the edge with a longer Euclidean length, store the other in PART[s][r]. Merge the faces adjacent to it.
    **end if**
  **end for**
**end for**

Now we analyze the time and space complexity of the part of our algorithm that construct the RMST.

It is clear that the operations polygon merging and edge deleting is done only once for each Delaunay edge. The polygon merging can be done in constant time if we use a hashed double-linked circular list to represent a polygon. So the whole BNG can be constructed in $O(m)$ time, where $m$ is the number of Delaunay edges. Since it is well known that the number of edges in a Delaunay triangulation is less than $3n$, where $n$ is the number of points, we can construct the BNG in $O(n)$ time if we already have the Delaunay triangulation. There exists some $O(n \log n)$ $L_1$ Delaunay triangulation algorithms [19], [20]. If we use one of them, the overall time complexity for the BNG construction is $O(n \log n)$. We need $O(n)$ space to record the shortest edges in the unique regions and the Delaunay triangulation algorithm also takes $O(n)$ space. So the overall space requirement is $O(n)$.

After the BNG is constructed, we can apply any MST algorithm on the resulting BNG to get an MST. Since the BNG is planar, we can apply an efficient $O(n)$ algorithm as described in [18] on it. Anyway, we get an $O(n \log n)$ time algorithm to compute the MST as a byproduct.

[3]One such data structure called the *double-linked edge list* can be used in this algorithm




(a) Three points in general position.

(b) Three points in general position.

Fig. 4. Comparison of visibility.

## IV. Polygonal Contraction

Now that we have the BNG and the RMST, we are going to improve the RMST into an RSMT using the neighborhood information contained in the BNG. In this section we describe the main part of our algorithm, the polygonal contraction method.

Our main idea is to find the visible triples in the polygonal faces of the BNG and contract them. A *visible triple* is a group of three points which can "see" each other without blocking by the edges of the BNG. By *contracting a triple* we mean to link a triple with a Steiner point and delete the longest edges in the cycles thus formed. In Fig. 4(a), the triple $\tau = (v_1, v_2, q)$ is a visible triple and in Fig. 4(b) they are contracted together. Contracting some triples will reduce the tree length—we gained length reduction from them—while others not. So our main idea is to find the triples with the positive gains and contract them. It is detailed below.

We examine each polygon of the BNG as follows. For each MST edge $e = (v_1, v_2)$ in the polygon, we determine the "best" vertex for the edge. We examine one by one the vertices of the polygon, excluding the end points of $e$. Let the vertex we are considering to be $q$ (see Fig. 4(a)), this will form a vertex-edge pair $(q, e)$. We consider the Steiner tree for the triple $\tau = (v_1, v_2, q)$. The Steiner point is at $c$

(See Fig. 4(b)). If we add $c$ to the point set and connect it to $v_1$, $v_2$ and $q$, two cycles will be formed. To get a new MST, we break the cycles by deleting the two longest edges $e_1$ and $e_2$ in each cycle. We denote $R(\tau) = \{e_1, e_2\}$ and call these two edges *bridges*. The *gain* of $\tau$ is $gain(\tau) = cost(\tau) - cost(R(\tau))$. If $gain > 0$, the addition of the point $c$ will reduce the tree length. Clearly in a series of contraction, each edge can only be used once as a bridge. There will be a "best" vertex with respect to edge $e$ that will lead to the largest gain. After this vertex is found, we record it together with the corresponding bridges and the edge $e$ in the set $T$. Then we examine the next edge in the polygon.

After the "best" vertex-edge pairs are extracted from the BNG. We sort them in descending order of their gains and start a batched contracting process starting from the most gained vertex-edge pair. If both the bridges associated with the pair are not marked yet, we add the Steiner point for the pair in to the point set, mark the bridges as *contracted*. Finally, we get the additional Steiner points in our point set.

This process can be summarized as in Algorithm 2.

**Algorithm 2** Polygonal Contraction

**Require:** The input is a BNG. The output is the Steiner points.
  **for** each polygonal face $p$ in the BNG **do**
    **for** each MST edge $e = (v_1, v_2)$ in $p$ **do**
      **for** each vertex $q$ other than $v_1$ and $v_2$ **do**
        find the bridges for the triple $\tau = (v_1, v_2, q)$
        $gain(\tau) = cost(\tau) - cost(R(\tau))$
        **if** $gain > 0$ **then**
          record $\tau$ together with $e$ and $R(\tau)$ in the set $T$
        **end if**
      **end for**
    **end for**
  **end for**
  sort the triples in $T$ in descending gains.
  **for** each triple $\tau$ in $T$ **do**
    **if** the neither of the bridges of $\tau$ is marked **then**
      add the Steiner point of $\tau$ into the set of Steiner points $S$ and mark the two bridges of $\tau$
    **end if**
  **end for**

To find the longest edge in a cycle, we can use the Hierarchical Greedy Preprocessing as proposed by Kahng et al. [13] during the MST construction phase. And then we can find the longest edge on the path of any two points in $O(\log n)$ time. Thus the time complexity for examine a polygon is $O(k^2 \log n)$, where $k$ is the number of vertices in the polygon. It takes $O(p \log n)$ time to get all the "best" vertex-edge pairs from the polygons of the BNG, where $p$ is the number of polygonal faces in the BNG. It can be proved that $p$ is linear with respect to $n$, so the polygonal contraction takes $O(n^2 \log n)$ in the worse case. But usually the maximum number of vertices in the polygons is small, so our algorithm runs faster than $O(n \log n)$ algorithms in practice (See Section VI).

After one phase of the polygonal contraction, we repeat the RMST construction process to get an RMST for the new point set and delete the Steiner points of degree 1 and 2. This new RMST is actually an RSMT for the original point set. We can repeat the polygonal contraction process to further reduce the length of the tree. Each iteration phase will reduce the tree length, but usually this process will gain less than 0.01% after 5 iterations, so we can stop iteration at this point.

We outline our heuristic Polygonal Contraction Heuristic (PCH) as a whole in Algorithm 3.

**Algorithm 3** Polygonal Contraction Heuristic (PCH)

**Require:** The input is a set of points. The output is a Steiner tree.
  **repeat**
    Construct the Delaunay triangulation on the point set.
    Construct the BNG as in Algorithm 1.
    Construct the RMST from the BNG.
    Do polygonal contraction as in Algorithm 2.
  **until** the MST can not be improved much

## V. Proof of Algorithm Correctness

Now we will prove that the BNG contains at least one MST, the guarantee for the correctness of our algorithm.

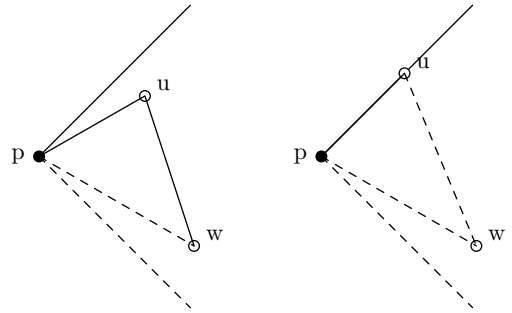
By the cycle property of the MST [21], we can safely delete the longest edge from a cycle. In Fig. 5(a) for example, $pw$ is the longest edge in the triangular cycle $pwu$(according to the weak uniqueness property) and can be deleted safely without destroying the MST in the graph. To prove the correctness of our algorithm, it suffices to argue that our algorithm can delete only one edge from each triangular cycle.

Only one question remains. Our algorithm doesn't work with cycles, but with points. We iterate from point to point, checking the length of the incident edges and deleteing the long ones. Will there be chances that we delete two different edges from the same triagular cycle when we are working on two different points(see Fig. 5(b))? If it is possible, we might isolate a point from the graph.

Now we will show this is not possible.

*Theorem 2:* The uniqueness pruning process cannot isolate a point from a triangle in any connected graph.

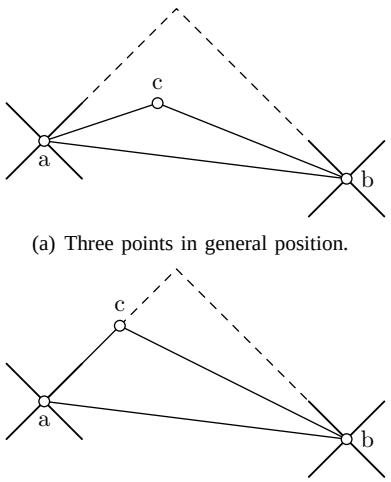
*Proof:* To isolate a point from a triangle, at least two edges must be deleted. But we will show that the pruning process can delete only one edge from a triangle. Consider the situation depicted in Fig. 6(a). First we assume that $ac$ and $bc$ are both in general position, that is, they are not on a half-line. So they cannot be of equal length as $ab$. Without loss of generality, we assume that $ac$ and $ab$ lie in




(a) Deleting an edge safely.

(b) Two edges deleted from the same triangle. We delete $uw$ when we check $u$ and we delete $pw$ when we check $p$.

Fig. 5. Can a point be isolated?




(a) Three points in general position.

(b) Three points with $\|ba\| = \|bc\|$.

Fig. 6. Two conflicts in one triangle.

some weak diagonal region with respect to $a$, and $bc$ and $ba$ lie in some weak diagonal region with respect to $b$. Here is two conflicts in the triangle $\triangle abc$. Note that the angle $\angle acb$ is larger than 90 degrees and edges $ca$ and $cb$ cannot be both inside any weak diagonal region. So there can be at most two conflicts and $ab$ must be the longest edge in the triangle $\triangle abc$. In both conflicts, only $ab$ will be considered for deletion by the pruning process and no points will be isolated.

Now we consider the case of ties, as in Fig. 6(b). We have $\|ba\| = \|bc\|$. If we choose to delete $bc$ when pruning on $b$, later the process would delete $ab$ when pruning on $a$, and $b$ will be isolated. So we must choose to delete $ba$ when pruning on $b$. Remember that in the pruning process, we choose to delete the edge opposite to the obtuse angle in case of ties. So we will choose to delete $ab$, and no points can be isolated. ■

Now the correctness of our algorithm is established.

## VI. Experimental Results and Discussions

We have implemented the algorithm using the C programming language. We compare our algorithm with several other heuristics by input from random points generated by the `gen` program that comes with the GeoSteiner package, which generates points on a 1000000 $\times$ 1000000 grid. All programs are compiled with the GCC compiler with same optimization options and run on the same machine, a Sun Fire 880 with 8G RAM.

We compare their performance and running time in Table I. The first column is the $O(n^4 \log n)$ Iterated 1-Steiner heuristic (I1S) proposed by Kahng and Robins [22]. This version is implemented by David M. Warme that come with the GeoSteiner 3.1 package.[4] The second column is a modified version of our heuristic that improves the MST by contracting the triangular faces of the Delaunay triangulation. The third column is the $O(n \log^2 n)$ BGA algorithm proposed by Kahng et al [13], which is a fast heuristic that has a slightly better performance than the I1S. The fourth column is the spanning graph based $O(n \log n)$ heuristic by Zhou [14]. The fifth column is our polygonal contraction heuristic (PCH). The last column is results from GeoSteiner 3.1 [9], which uses FST pruning and branch-and-bound techniques. For point sets larger than 2000, GeoSteiner with the `lp_solve` package failed to produce results, which is considered to be a bug of `lp_solve`. Note that the timing results for small input sets may be inaccurate due to the system timer's resolution. The last rows are results from large random inputs, but in practice we seldom meet input sets larger than 30000. We list them there only to illustrate the growth of running time of our algorithm.

The results show that the performance of our heuristic matches or exceeds other algorithms yet much faster. Our algorithm's better performance than the Delaunay approach and Zhou's spanning graph based algorithm indicates that the BNG polygonal faces contains better neighborhood information. The comparison of the running time between Zhou's algorithm and ours shows that although the worst case complexity is $O(n^2 \log n)$, our algorithm has better practical running time and is more stable according to various inputs.

## VII. Conclusions

We have proposed an integrated heuristic for Steiner tree construction based on the BNG and the idea of polygonal contraction. The heuristic has a performance comparable to the best RSMT heuristics yet it runs much faster in practice. We are currently extending the heuristic to handle other practical considerations, such as routing obstacle, preferred direction and via costs.

[4]We didn't use the $O(n^3)$ B1IS implimentation by Robins, because his program doesn't accept streamed input from other programs. Thus it cannot be used for comparison.

TABLE I

COMPARISON OF THE PERFORMANCE AND RUNNING TIME OF SEVERAL OTHER HEURISTICS AND THE POLYGONAL CONTRACTION HEURISTIC. (AVERAGED OVER 20 RUNS)

| Input | I1S | | Delaunay based | | BGA | | Spanning Graph | | PCH | | GeoSteiner 3.1 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | %Imp | CPU | %Imp | CPU | %Imp | CPU | %Imp | CPU | %Imp | CPU | %Imp | CPU |
| 100 | 10.91 | 37.88 | 9.20 | 0.02 | 11.04 | 0.01 | 9.80 | 6.83 | 11.03 | 0.01 | 11.34 | 0.6 |
| 200 | 10.95 | 666.97 | 9.65 | 0.02 | 11.05 | 0.04 | 10.64 | 12.19 | 11.16 | 0.04 | 11.39 | 5.75 |
| 400 | - | $> 6000$ | 9.47 | 0.07 | 10.87 | 0.09 | 10.47 | 9.71 | 10.84 | 0.10 | 11.32 | 45.14 |
| 800 | - | - | 9.62 | 0.18 | 10.96 | 0.28 | 10.61 | 10.7 | 10.79 | 0.21 | 11.67 | 106.88 |
| 1000 | - | - | 9.36 | 0.22 | 11.04 | 0.31 | 10.67 | 18.3 | 10.90 | 0.30 | 11.49 | 199.45 |
| 2000 | - | - | 9.45 | 0.63 | 11.05 | 1.03 | 10.62 | 18.92 | 10.95 | 0.70 | - | - |
| 4000 | - | - | 9.89 | 1.08 | 11.02 | 2.43 | 10.52 | 20.10 | 10.95 | 1.98 | - | - |
| 8000 | - | - | 9.87 | 2.87 | 11.01 | 8.34 | 10.50 | 23.53 | 10.98 | 3.39 | - | - |
| 10000 | - | - | 9.88 | 2.98 | 11.01 | 13.09 | 10.58 | 24.20 | 11.01 | 4.08 | - | - |
| 20000 | - | - | 9.90 | 6.43 | 11.02 | 38.58 | 10.48 | 30.52 | 10.98 | 8.84 | - | - |
| 40000 | - | - | 9.86 | 19.62 | 11.04 | 85.83 | 10.53 | 44.52 | 11.01 | 18.49 | - | - |

## VIII. ACKNOWLEDGEMENTS

The authors wish to thank Prof. Hai Zhou in Northwestern University for providing us the program of the spanning graph based algorithm. Thanks also go to the anonymous reviewers for their helpful comments.

## REFERENCES

[1] Yu Hu, Tong Jing, Xianlong Hong, Zhe Feng, Xiaodong Hu, and Guiying Yan. An efficient rectilinear steiner minimum tree algorithm based on ant colony optimization. In *Proceedings of IEEE ICCCAS*, pages 1276–1280, Chengdu, China, 2004.

[2] Qi Zhu, Hai Zhou, Tong Jing, Xianlong Hong, and Yang Yang. Efficient octilinear steiner tree construction based on spanning graphs. In *Proceedings of IEEE/ACM ASP-DAC*, pages 687–690, Yokohama, Japan, 2004.

[3] Yukun Cheng, Guiying Yan, and Tong Jing. The structural properties of hexagonal steiner minimum trees for terminals on the boundary of an equilateral triangle. In *Proceedings of ISC&I*, pages 61–65, Zhuhai, China, 2004.

[4] Jingyu Xu, Xianlong Hong, Tong Jing, Yici Cai, and Jun Gu. An efficient hierarchical timing-driven steiner tree algorithm for global routing. *INTEGRATION, the VLSI J*, 35(2):69–84, 2003.

[5] H. Y. Bao, X. L. Hong, and Y. C. Cai. Timing-driven steiner tree algorithm based on sakurai model. *Chinese Journal of Semiconductors*, 20(1):41–46, 1999.

[6] Yu Hu, Zhe Feng, Tong Jing, Xianlong Hong, Yang Yang, Ge Yu, Xiaodong Hu, and Guiying Yan. Forst: A 3-step heuristic for obstacle-avoiding rectilinear steiner minimal tree construction. In *Proceedings of ISC&I*, pages 1017–1021, Zhuhai, China, 2004.

[7] Yang Yang, Qi Zhu, Tong Jing, Xianlong Hong, and Yin Wang. Rectilinear steiner minimal tree among obstacles. In *Proceedings of IEEE ASICON*, pages 348–351, Beijing, China, 2003.

[8] M. R. Garey and D. S. Johnson. The rectilinear steiner problem is *NP-Complete*. *SIAM Journal on Applied Mathematics*, 32:826–834, 1977.

[9] D. M. Warme, P. Winter, and M. Zacharisen. Geosteiner 3.1. Package.

[10] Sanjeev Arora. Polynomial time approximation schemes for the euclidean TSP and other geometric problems. In *IEEE Symposium on Foundations of Computer Science*, pages 2–11, 1996.

[11] Jeff Griffith, Gabriel Robins, Jeffrey S. Salowe, and Tongtong Zhang. Closing the gap: Near-optimal steiner trees in polynomial time. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 13(11):1351–1365, November 1994.

[12] M. Borah, R. M. Owens, and M. J. Irwin. An edge-based heuristic for steiner routing. *IEEE Transactions on Computer Aided Design*, 13(1563–1568), 1994.

[13] Andrew B. Kahng, Ion I. Măndoiu, and Alexander Z. Zelikovsky. Highly scalable algorithms for rectilinear and octilinear steiner trees. In *Proc. Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 827–833, 2003.

[14] Hai Zhou. Efficient steiner tree construction based on spanning graphs. In *Proceedings of ISPD'03*, April 2003.

[15] Andew Chi-Chih Yao. On constructing minimum spanning trees in $k$-dimensional spaces and related problems. *SIAM Journal on Computing*, 11(4):721–736, November 1982.

[16] Leonidas J. Guibas and Jorge Stolfi. On computing all north-east nearest neighbors in the $L_1$ metric. *Information Processing Letters*, 17, 1983.

[17] Hai Zhou, N. Shenoy, and W. Nicholls. Efficient spanning tree construction without delaunay triangulation. *Information Processing Letters*, 81(5), 2002.

[18] David Cheriton and Robert Endre Tarjan. Finding minimum spanning trees. *SIAM Journal on Computing*, 5(4):724–742, December 1976.

[19] D. T. Lee and C. K. Wong. Voronoi diagrams in $L_1(L_\infty)$ metric with 2-dimensional storage applications. *SIAM Journal of Computing*, 9:200–211, 1980.

[20] Gary M. Shute, Linda L. Deneen, and Clark D. Thomborson. An $O(n \log n)$ plane-sweep algorithm for $L_1$ and $L_\infty$ delaunay triangulations. *Algorithmica*, 6:207–221, 1991.

[21] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorihtms*. The MIT Press, Cambridge, MA, 1990.

[22] A. B. Kahng and G. Robins. A new class of iterated steiner tree heuristics with good performance. *IEEE trans. Computer-Aided Design*, 11:893–902, 1992.