Efficient Minimum Spanning Tree Construction without Delaunay Triangulation

Hai Zhou, Narendra Shenoy, and William Nicholls
Advanced Technology Group
Synopsys, Inc.
Mountain View, CA 94043

Abstract

Minimum spanning tree problem is a very important problem in VLSI CAD. Given n points in a plane, a minimum spanning tree is a set of edges which connects all the points and has a minimum total length. A naive approach enumerates edges on all pairs of points and takes at least $\Omega(n^2)$ time. More efficient approaches find a minimum spanning tree only among edges in the Delaunay triangulation of the points. However, Delaunay triangulation is not well defined in rectilinear distance. In this paper, we first establish a framework for minimum spanning tree construction which is based on a general concept of spanning graphs. A spanning graph is a natural definition and not necessarily a Delaunay triangulation. Based on this framework, we then design an $O(n \log n)$ sweep-line algorithm to construct a rectilinear minimum spanning tree without using Delaunay triangulation.

1 Introduction

Minimum spanning tree problem has wide applications in VLSI CAD. Given n points in a plane, a minimum spanning tree is a set of edges which connects all the points and has a minimum total length. It is frequently used as a metric of wire length estimation during placement. It is often constructed to approximate a minimum Steiner tree and is also a key step in many Steiner tree heuristics. It is also used in an approximation to the traveling salesperson problem which can be used to generate scan chains in testing. In the application which motivated our study, our interest lies in using the minimum spanning tree to construct a set of searches defined by the edges, to route an n pin net. It is important to emphasize that for real world applications, often n will be large. The spanning tree is an abstract model for a route of a net in our application. Since the problem formulation deals with points and most physical data have non-zero dimensions, a much more realistic model can be achieved by representing "large" objects as sets of points. Typically for a big net with large pre-routes, this can easily exceed 100,000 points. Since it is a problem which will be computed hundreds of thousands times and many of them will have very large input sizes, the minimum spanning tree problem definitely deserves a very efficient solution.

Minimum spanning tree construction on an arbitrary graph is a well studied problem [1]. It also belongs to a more general class of greedy problems on combinatorial structures known as matroids [6]. Typical complexity of computing a minimum spanning tree in a graph $G(V, E)$ is $O(m \log n)$, where n is the number of vertices and m is the number of edges. So given a graph, the spanning tree can be constructed efficiently. Clearly, a minimal spanning tree is contained in the complete graph of n points. However enumerating $\Omega(n^2)$ edges is expensive for large n.

The first algorithm to speed up the minimum spanning tree computation came as a by-product of computational geometry research and was based on the fact that only edges in the Delaunay triangulation of the points need to be examined [8]. But this only works in Euclidean distance (L_2). When Euclidean distance (L_2) is used, the Delaunay graph is defined as the dual of the Voronoi diagram [8]. A Delaunay graph is usually a triangulation if no more than three points are cocircular, or can be made a triangulation by adding more edges. Their algorithm [8] for Voronoi diagram uses divide-and-conquer strategy and runs in $O(n \log n)$ time. Later, Fortune [2] designed a much simpler sweep-line algorithm with the same running time. His algorithm avoids the difficult merge step of the divide-and-conquer technique. However, when rectilinear distance (L_1) is used, the Voronoi diagram is not always well defined. Efforts to resolve this issue need to explicitly specify what they mean when there are ambiguities [5, 3]. Similar as in Euclidean distance, the original algorithm [5] in this direction was a divide-and-conquer algorithm. And motived by Fortune [2], there came a sweep-line algorithm [3] more recently.

As we mentioned early, minimum spanning tree construction for Euclidean distance came only as a by-product of Delaunay triangulation. Since Delaunay triangulation is not well defined in rectilinear distance, forcing minimum spanning tree computation on it encounters unnecessary difficulties. In fact, parallel to Delaunay triangulation, Yao [11] observed that a minimum spanning tree can be constructed by considering a sufficient number of closest neighbors for each of the given points and gave an algorithm which runs in $O(n^{2-1/8}\log^{1-1/8} n)$ time for the planar case. Guibas and Stolfi [4] further implemented the idea for rectilinear distance in the plane with a running time of $O(n \log n)$. Interesting enough, their algorithm is also based on divide-and-conquer strategy: it divides the point set into a left half and a right half, and recursively applies the algorithm to them.

In this paper, we focus directly on the objective of constructing a minimum spanning tree. Keeping this in mind, we find that there is no need to take the burden of constructing or even defining a Delaunay triangulation. Actually, what we need are sparse graphs which contains minimum spanning trees. Generally, we define these graphs as spanning graphs. Although for Euclidean distance a Delaunay triangulation can be proved to be a spanning graph, a spanning graph need not to be a Delaunay triangulation. This observation is invaluable for rectilinear distance metric where a Delaunay triangulation is not well defined. Based on the framework and using the property that each points needs to be connected to only a few other points, we designed a sweep-line algorithm to construct a spanning graph for rectilinear distance. After that, a minimum spanning tree can be easily computed on the spanning graph.

With respect to the literature, our work stands out on two contributions: First, we establish a general framework of spanning graphs which includes both Delaunay triangulation and non-Delaunay-triangulation approaches, and study the properties of spanning graphs in both Rectilinear and Euclidean distances. Second, although the divide-and-conquer algorithm by Guibas and Stolfi [4] has the same asymptotic running time as our sweep-line algorithm, theirs is more complicated in implementation, requires more storage ($O(n \log n)$ vs. $O(n)$), and has larger hiding constant.

The rest of the paper is organized as follows. In Section 2, we define the spanning graphs and discuss their properties. In Section 3, we design an algorithm to construct rectilinear spanning graphs for a given set of points. Finally, Section 4 gives some experimental results and Section 5 concludes the paper.

2 Spanning Graph

Given a set of n points in a plane, a spanning tree is a set of edges that connects all the points and contains no cycles. When each edge is weighted using some distance metric of the incident points, the *metric minimum spanning tree* is a tree whose sum of edge weights is minimum. If the Euclidean distance (L_2) is used, it is called the *Euclidean minimum spanning tree*; if the rectilinear distance (L_1) is used, it is called the *rectilinear minimum spanning tree*. Since the minimum spanning tree problem on a weighted graph is well studied, the usual approach for metric minimum spanning tree is to first define an weighted graph on the set of points and then to construct a spanning tree on it.

Much like a connection graph is defined for the maze search [12], we can define a spanning graph for the minimum spanning tree construction.

Definition 1 *Given a set of points V, an undirected graph $G = (V, E)$ is called a* spanning graph *if it contains a minimum spanning tree.*

Usually, for a given set of points, the minimum spanning tree may not be unique. Thus a spanning graph defined above may not contain all minimum spanning trees. If we are only interested in one of these trees, no matter which one, the above definition is sufficient. Otherwise, we may need a strong version as follows.

Definition 2 *Given a set of points V, an undirected graph $G = (V, E)$ is called a* strong spanning graph *if it contains all minimum spanning trees.*

Since we are interested in spanning graphs with as few number of edges as possible, we define the *cardinality* of a spanning graph as its number of edges. As we can see, a complete graph on a set of points contains all spanning trees, thus is a spanning graph, even in the strong sense. This gives us an $O(n^2)$ upper bound on the cardinalities of both spanning graphs and strong spanning graphs. On the other hand, since a minimum spanning tree is by definition also a spanning graph, the minimum cardinality of spanning graphs is always $n-1$ for a set of n points. But the minimum cardinality of strong spanning graphs is more complicated and, as we will show next, is dependent on which metric is used.

Minimum spanning tree algorithms usually use two properties to infer the inclusion and exclusion of edges in a minimum spanning tree. The first property is known as the *cut property*. It states that an edge of smallest weight crossing any partition of the vertex set into two parts belongs to a minimum spanning tree. The second property is known as the *cycle property*. It says that an edge with largest weight in any cycle in the graph can be safely deleted. Since the two properties are stated in connection with the construction of a minimum spanning tree, they are related to a spanning graph. A strong cut property states that all lightest edges crossing any partition of the vertex set into two parts belong to a strong spanning graph. A strong cycle property says that the single heaviest edge in any cycle in the graph does not belong to a strong spanning graph. Preparata and Shamos [8] proved the following lemma.

Lemma 1 (Lemma 6.2 in [8]) *Let S be a set of points in the plane, and let $\triangle(p)$ denote the set of points adjacent to $p \in S$ in the Delaunay triangulation of S. For any partition $\{S_1, S_2\}$ of S, if $\overline{qp}$ is the shortest segment between points of S_1 and points of S_2, then q belongs to $\triangle(p)$.*

Combining the lemma with the strong cut property, we have the following theorem.

Theorem 1 *If Euclidean distance is used, the Delaunay triangulation of a set of points is always a strong spanning graph.*

Since a Delaunay graph has only linear number of edges, the above theorem also shows that the minimum cardinality of strong spanning graph is $O(n)$ for any set of n points if Euclidean distance is used. On the contrary, the rectilinear distance does not have such good property, as shown in the following lemma.

Theorem 2 *If rectilinear distance is used, there is a set of n points, for which any strong spanning graph has at least $\Omega(n^2)$ number of edges.*

Proof: Consider a set of n points as follows. Let $\lceil n/2 \rceil$ points fall on the segment $x + y = 1$ with $x \in [0, 1]$; all other points sit on the segment $x + y = -1$ with $x \in [-1, 0]$. This is illustrated in Figure 2. As we can see, if we partition the whole set into two subsets according to the two segments, all edges between the two subsets have the same length. According to the strong cut property, they all must be in a strong spanning graph. □

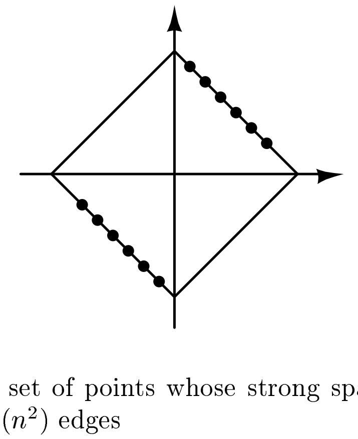

Figure 1: A set of points whose strong spanning graph must have $\Omega(n^2)$ edges

3 Rectilinear Spanning Graph Construction

Using the terminology given in [10], we define the *uniqueness property* as follows.

Definition 3 *Given a point s, a region R has the* uniqueness property *with respect to s if for every pair of points $p, q \in R$, $||pq|| < \max(||sp||, ||sq||)$. A partition of space into a finite set of disjoint regions is said to have the uniqueness property if each of its regions has the uniqueness property.*

For the rest of the paper we will use notation $||sp||$ to represent the distance between s and p using the L_1 metric. Define the *octal partition* of the plane with respect to s as the partition induced by the two rectilinear lines and the two 45 degree lines through s, as shown in Figure 3(a). Here, each of the regions R_1 through R_8 includes only one of its two bounding half line as shown in Figure 3(b). It can be shown that the octal partition has the uniqueness property.

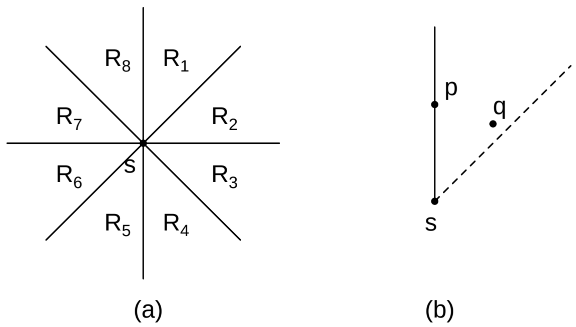

Figure 2: Octal partition and the uniqueness property

Lemma 2 *Given a point s in the plane, the octal partition with respect to s has the uniqueness property.*

Proof: To show a partition has the uniqueness property, we need to prove that each region of the partition has the uniqueness property. Since the regions R_1 through R_8 are similar to each other, we only give a proof for R_1.

The points in R_1 can be characterized by the following inequalities

$$x \geq x_s,$$
$$x - y < x_s - y_s.$$

Suppose we have two points p and q in R_1. Without loss of generality, we can assume $x_p \leq x_q$. If $y_p \leq y_q$, then we have $||sq|| = ||sp|| + ||pq|| > ||pq||$. Therefore we only need to consider the case when $y_p > y_q$. In this case, we have

$$\begin{aligned}
||pq|| &= |x_p - x_q| + |y_p - y_q| \\
&= x_q - x_p + y_p - y_q \\
&= (x_q - y_q) + y_p - x_p \\
&< (x_s - y_s) + y_p - x_s \\
&= y_p - y_s \\
&\leq x_p - x_s + y_p - y_s \\
&= ||sp||
\end{aligned}$$

Given two points p, q in the same octal region of point s, the uniqueness property says that $||pq|| < \max(||sp||, ||sq||)$. Consider the cycle on points s, p, and q. Based on the cycle property, only the point with the minimum distance from s needs to be connected to s. An interesting property of the octal partition is that the contour of equi-distant points from s forms a line segment in each region . In regions R_1, R_2, R_5, R_6, these segments are captured by an equation of the form $x + y = c$; in regions R_3, R_4, R_7, R_8, they are described by the form $x - y = c$, as shown in Figure 3.

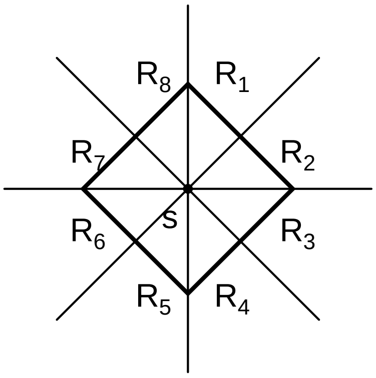

Figure 3: Equal distance points in each region

Conceptually, we only need to consider edges from s to the closest neighbor in each octant. We will pose this problem in the reverse manner. Given a point s, find all the candidate points to which it can possibly be the nearest neighbor in a specified octant. For sake of simplifying the exposition, we will only consider the case for R_1. The rest of octants are symmetric and the discussion can be easily extended to handle them. For the R_1 octant, we construct a sweep line algorithm on all points according to non-decreasing $x + y$. During the sweep, we maintain an *active* set consisting of points whose nearest neighbors in R_1 are still to be discovered. When we process a point p, we find all the points in the active set, which have p in their R_1 regions. Suppose s is one such point from the active set. Since we process points in non-decreasing $x + y$, we know that p is the nearest point in R_1 for s. Therefore, we add edge sp and delete s from the active set. After processing those active points, we also add p into the active set. Each point will be added and deleted at most once from the active set.

The fundamental operation that is required in the sweep line algorithm is given a point p, find a subset of active points such that p is in their R_1 regions. Based on the following observation, we need to find the subset of active points in the R_5 region of p.

Observation 1 *Given two points p and s, point p is in the R_1 region of s if and only if s is in the R_5 region of p.*

Since R_5 can be represented as a two-dimensional range $(-\infty, x_p] \times (x_p - y_p, +\infty)$ on $(x, x - y)$, a priority search tree [7] can be used to maintain the active point set. Since each of the insertion and deletion operations takes $O(\log n)$ time, and the query operation takes $O(\log n + k)$ time where k is the number of objects within the range, the total time for the sweep is $O(n \log n)$. Since other regions can be processed in the similar way as in R_1, we get an algorithm running in $O(n \log n)$ time. Priority search tree is a data structure that relies on maintaining a balanced structure for the fast query time. This works well for static input sets. When the input set is dynamic, rebalancing the tree can be quite challenging. Fortunately, the active set has a structure we can exploit for an alternate representation. Since we delete a point from the active set if we find a point in its R_1 region, no point in the active set can be in the R_1 region of another point in the set.

Lemma 3 *For any two points p, q in the active set, we have $x_p \neq x_q$, and if $x_p < x_q$ then $x_p - y_p \leq x_q - y_q$.*

Based on this property, we can order the active set in increasing order of x. This implies a non-decreasing order on $x - y$. Given a point s, the points which have s in their R_1 region must obey the following inequalities

$$x \leq x_s,$$

$$x - y < x_s - y_s.$$

To find the subset of active points which have s in their R_1 regions, we can first find the largest x such that $x \leq x_s$, then proceed in decreasing order of x until $x - y \geq x_s - y_s$. Since the ordering is kept on only one dimension, using any binary search tree with $O(\log n)$ insertion, deletion, and query time will also give us an $O(n \log n)$ time algorithm. Binary search trees also need to be balanced. An alternative is to use skip-lists [9] which use randomization to avoid the problem of explicit balancing but provide $O(\log n)$ expected behavior.

A careful study also shows that after the sweep process for R_1, there is no need to do the sweep for R_5, since all edges needed in that phase are either connected or implied. This is also based on Observation 1. Moreover, based on the information in R_5, we can further reduce the number of edge connections. As shown in Figure 3, when the sweep step processes point s, we find a subset of active points which have s in their R_1 regions. Without lost of generality, suppose p and q are two of them. Then p and q are in the R_5 region of s, which means $||pq|| < \max(||sp||, ||sq||)$. Therefore, we need only to connect s with the nearest active point.

Since R_1 and R_2 have the same sweep sequence, we can process them together in one pass. Similarly, R_3 and R_4 can be processed together in another pass. Based on

```
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 the nearest point in each subset;
      delete the subsets from A[1], A[2], respectively;
      add p to A[1], A[2];
    }
  }
```

Figure 5: The rectilinear spanning graph algorithm

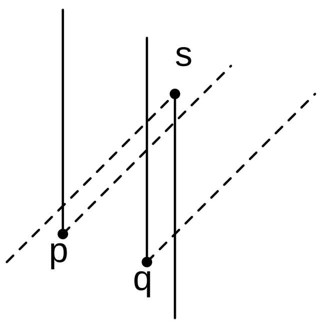

Figure 4: Only the nearest point in R_5 need to be connected

the above discussion, the pseudo-code of the algorithm is presented in Figure 5.

The correctness of the algorithm is stated in the following theorem.

Theorem 3 *Given n points in the plane, the rectilinear spanning graph algorithm constructs a spanning graph in $O(n \log n)$ time, and the number of edges in the graph is $O(n)$.*

Proof: The algorithm can be considered as deleting edges from the complete graph. As described, all edges that we delete are redundant based on the cycle property. Thus, the output graph of the algorithm will contain at least one rectilinear minimum spanning tree.

In the algorithm, each given point will be inserted and deleted at most once from the active set for each of the four regions R_1 through R_4. For each insertion or deletion, the algorithm requires $O(\log n)$ time. Thus, the total time is upper bounded by $O(n \log n)$. The storage we need is only for active sets, which is at most $O(n)$. □

4 Experimental Results

We implemented our Rectilinear Spanning Graph (RSG) algorithm in C language. Given a set of points, it constructs a weighted graph which contains at least one rectilinear minimum spanning tree. We then use Kruskal's algorithm to compute a minimum spanning tree on the graph. Since our algorithm is proven to be exact, we are only concerned about its running time. We compare our approach with two other approaches. The first approach uses the complete graph on the point set as the input to Kruskal's algorithm. The second approach is an implementation of concepts described in [10]; namely for each point, scan all other points but only connect the nearest one in each quadrant region. We generate input data using randomly generated points, with sizes ranging from 1000 to 20000. We report results of each approach in Table 1. The first column gives the number of generated points; the second column gives the number of distinct points. For each approach, we report the number of edges in the given graph and the total running time. For input size larger than 10000, the complete graph approach simply runs out of memory.

The proposed RSG algorithm results in fewer edges in the spanning graph, thereby resulting in faster overall performance. This becomes evident for large n.

5 Conclusion

In summary, we have characterized a broader class of spanning graphs which are more natural than the Delaunay triangulation for minimum spanning tree constructions. We also described a new and much simpler approach to solving the minimum spanning tree problem for the rectilinear distance metric. The approach is relies

Input		Complete		Bound-degree		RSG	
orig	distinct	#edge	time	#edge	time	#edge	time
1000	999	498501	5.095s	3878	0.299s	2571	0.112s
2000	1996	1991010	24.096s	7825	0.996s	5158	0.218s
4000	3995	7978015	2m7.233s	15761	3.452s	10416	0.337s
6000	5991	17943045	5m54.697s	23704	7.515s	15730	0.503s
8000	7981	31844190	13m7.682s	31624	13.141s	21149	0.672s
10000	9962	49615741	–	39510	20.135s	26332	0.934s
12000	11948	–	–	47424	32.300s	31586	1.052s
14000	13914	–	–	55251	46.842s	36853	1.322s
16000	15883	–	–	63089	1m3.759s	42251	1.486s
18000	17837	–	–	70876	1m19.812s	47511	1.701s
20000	19805	–	–	78723	1m45.792s	52732	1.907s

Table 1: Experimental Results

on the cycle property of spanning trees and the uniqueness property applied to an octal partition of the planar space.

References

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

[2] Steven Fortune. A sweepline algorithm for voronoi diagrams. *Algorithmica*, 2:153–174, 1987.

[3] Linda L. Deneen Gary M. Shute and Clark D. Thomborson. An $O(n \log n)$ Plane-Sweep Algorithm for L_1 and L_∞ Delaunay Triangulation. In *Algorithmica*, volume 6, pages 207–221, 1991.

[4] Leo J. Guibas and Jorge Stolfi. On computing all north-east nearest neighbors in the L_1 metric. *Information Processing Letters*, 17(4):219–223, 8 November 1983.

[5] F. K. Hwang. An $o(n \log n)$ algorithm for rectilinear minimal spanning trees. *Journal of the ACM*, 26(2):177–182, April 1979.

[6] E. L. Lawler. *Combinatorial Optimization: Networks and Matroids.* Holt, Rinehart and Winston, 1976.

[7] Edward M. McCreight. Priority search trees. *SIAM Journal of Computing*, 14(2):257–276, May 1985.

[8] Franco P. Preparata and Michael Ian Shamos. *Computational Geometry: An Introduction.* Springer-Verlag, 1985.

[9] W. Pugh. Skip lists: A probabilistic alternative to balanced trees. *Communications of the ACM*, 33(6), 1990.

[10] Gabriel Robins and Jeffrey S. Salowe. Low-degree minimum spanning tree. *Discrete and Computational Geometry*, 14:151–165, September 1995.

[11] Andrew 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.

[12] S. Q. Zheng, J. S. Lim, and S. S. Iyengar. Finding obstacle-avoiding shortest paths using implicit connection graphs. *IEEE Transactions on Computer Aided Design*, 15(1):103–110, January 1996.