BIT 26 (1986), 7–16

AN $O(N \log N)$ MINIMAL SPANNING TREE ALGORITHM FOR N POINTS IN THE PLANE *

R. C. CHANG and R. C. T. LEE

National Chiao Tung University, Hsinchu, Taiwan, Republic of China

National Tsing Hua University, Hsinchu, and Academia Sinica, Taipei, Taiwan, Republic of China

Abstract.

We shall present a divide-and-conquer algorithm to construct minimal spanning trees out of a set of points in two dimensions. This algorithm is based upon the concept of Voronoi diagrams. If implemented in parallel, its time complexity is $O(N)$ and it requires $O(\log N)$ processors where N is the number of input points.

1. Introduction.

We are concerned with algorithms to construct minimal spanning trees. The minimal spanning tree problem can be roughly stated as follows. Given a set of points, connect them into a tree such that its total length is minimized.

There are many algorithms to construct minimal spanning trees. For most algorithms to construct minimal spanning trees, the input data consist of a graph. We shall assume that our data are a set of points in two dimensions. It is well known that for a graph of N vertices, a minimal spanning tree (MST) can be found in $O(N^2)$ time. Bentley and Friedman [3] gave two algorithms for constructing minimal spanning trees in Euclidean space and showed that on the average, an MST can be constructed in time $O(N \log N)$. Nevalainen, Ernvall and Katajainen [9] proposed an improved version of Bentley and Friedman's MST algorithm. Katajainen [7] proposed an improved version of Bentley and Friedman's MST algorithm and showed that in the worst case the time complexity of Bentley and Friedman's algorithm is $O(N^2 \log N)$. Shamos and Hoey [11] proved that the minimal spanning tree in the plane can be extracted from a Voronoi diagram [14, 11]. A Voronoi diagram of 14 points is

Received February, 1985. Revised August 1985.

* This research was partially supported by the National Science Council of the Republic of China under the Grant NSC74-0408-E007-01.

shown in Fig. 1-1. The worst case time complexity of Shamos and Hoey's MST algorithm is $O(N \log N)$. Yao [15] proposed an algorithm for finding an MST in the k-dimensional space. By employing fast nearest neighbor searching algorithms, in the worst case, an MST can be found in time $O(N^{2-a(k)}(\log N)^{1-a(k)})$, where $a(k) = 2^{-(k-1)}$. This bound can be improved to $O((N \log N)^{1.8})$ for points in 3-dimensional space.

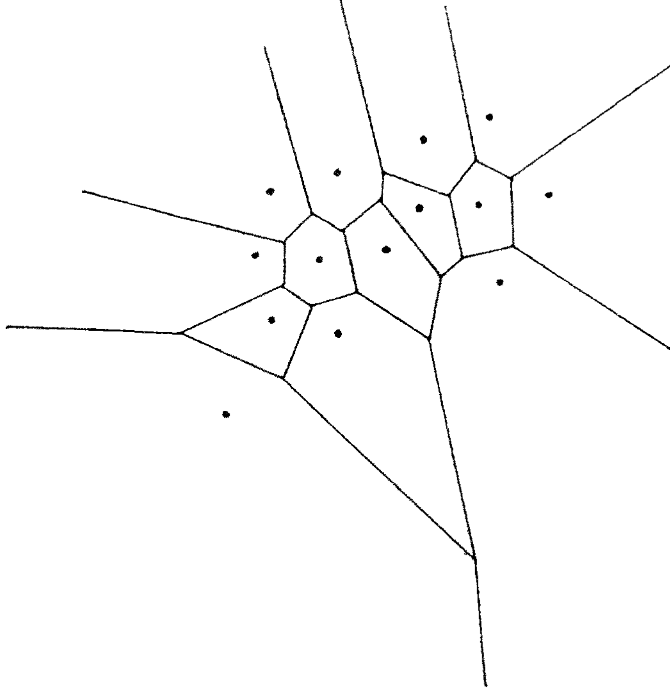

Fig. 1-1. A Voronoi diagram with 14 points.

Our research arises from one puzzling question: Can a minimal tree in a plane be formed by divide-and-conquer approach? Note that a minimal spanning tree connects a set of points into a tree. Since the total length of this tree is to be minimized, we expect two points to be connected if they are relatively close to each other. In other words, if there are several points located between two points P_1 and P_2, then it is quite unlikely that there will be an edge connecting P_1 and P_2 in a minimal spanning tree. This implies that the calculation of the distance between P_1 and P_2 should be avoided. Bentley and Friedman [4] discussed this point. They pointed out that in implementing a minimal spanning tree algorithm, if efficient nearest neighbor searching algorithms are used, we can avoid many distance calculations. For the case mentioned above where P_1 is relatively far away from P_2, we can probably avoid the distance calculation between them if a good nearest neighbor searching concept is incoporated into the minimal spanning tree construction algorithm, as it was in [3].

That a minimal spanning tree can be formed by using the nearest neighbor searching approach indicates that local information is crucial and divide-and-conquer approach may therefore be appropriate. In the following section we shall show that a minimal spanning tree can be obtained as a Voronoi diagram is being constructed. Since this can be formed by a divide-and-conquer approach, a minimal spanning tree in a two-dimensional plane can be constructed by the same approach.

In section 2 we present the divide-and-conquer approach to construct minimal spanning trees and prove the correctness of the algorithm. Section 3 shows that the complexity of this algorithm is $O(N \log N)$ and can be implemented as a parallel algorithm with complexity $O(N)$ using $O(\log N)$ processors. Concluding remarks are given in section 4.

2. The divide-and-conquer approach to construct minimal spanning trees.

That a minimal spanning tree for a set of points in the Euclidean plane can be constructed by a divide-and-conquer approach is based upon the following observation. Suppose we use a straight line L to divide these points into V_L and V_R respectively. The trees are denoted $MST(V_L)$ and $MST(V_R)$. If we try to merge these two minimal spanning trees, what will happen? Obviously, the leftmost part of $MST(V_L)$ and the rightmost part of $MST(V_R)$ will hardly be affected by this merging and they will probably remain unchanged. The only parts which will be affected by this merging process are the central parts of $MST(V_L)$ and $MST(V_R)$. Thus, if there is a correct way to merge $MST(V_L)$ and $MST(V_R)$ efficiently, then a minimal spanning tree can be constructed by using a divide-and-conquer approach.

It turns out that there is indeed a merging process which is efficient. Note that a minimal spanning tree is a part of the Delaunay triangulation [6], as shown in [10]. To merge $MST(V_L)$ and $MST(V_R)$, we merely examine the Delaunay triangulation edges between $MST(V_L)$ and $MST(V_R)$ to see whether they should be added to connect these two minimal spanning trees. We shall show later that the algorithm to find Delaunay triangulations proposed in [8] can be used as the merging process because it will find all of the Delaunay edges between $MST(V_L)$ and $MST(V_R)$.

Since a minimal spanning tree is a connected graph, points in V_L must be connected to points in V_R. In other words, edges must be added. If more than one edge is added, then a cycle is formed and some edge in this cycle must be deleted to keep the resulting graph a tree.

Thus our merging process involves two kinds of actions: adding new edges and possibly deleting edges. Because all minimal spanning tree edges must be Delaunay edges [10], we only have to consider edges which are Delaunay edges. To determine whether an edge is a Delaunay edge or not, we may use the method proposed in [8]. This algorithm was originally designed for merging two Delaunay

triangulations. Here we may simply view it as an algorithm to find a candidate set of Delaunay edges.

As pointed out in [8], there is an ordering of candidate edges. The merging process adds the candidate edges one by one from bottom to top. If a cycle is formed after adding an edge, the longest edge in the cycle is deleted. For the minimal spanning trees in Fig. 2-1, (v_3, v_{10}) is the first edge to be added. After (v_6, v_8) is added, a cycle is formed. In this cycle, (v_3, v_{10}) is the longest and is thus deleted. Finally, (v_5, v_7) is added. A cycle is again formed, and we find that (v_5, v_6) is the longest in this cycle and so it is deleted. The resulting minimal spanning tree is shown in Fig. 2-2.

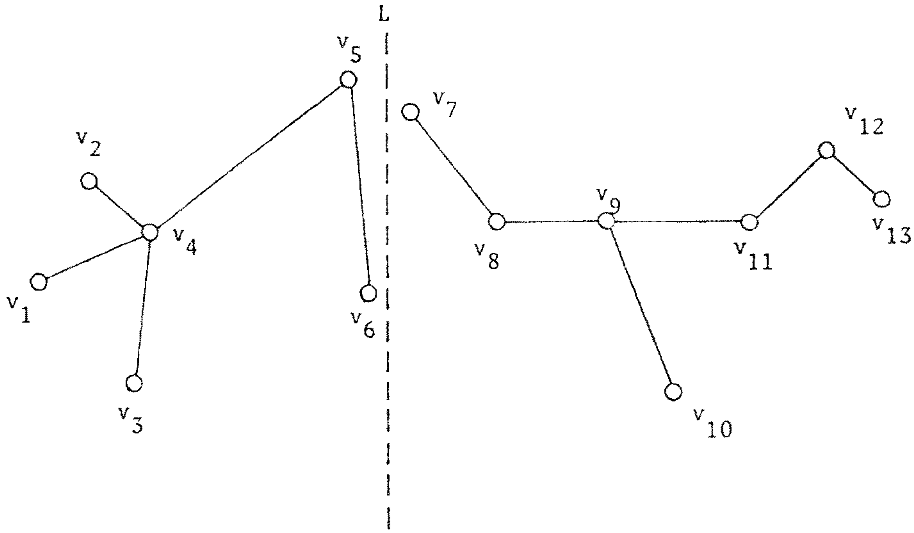

Fig. 2-1. Two *MST*'s which are to be merged.

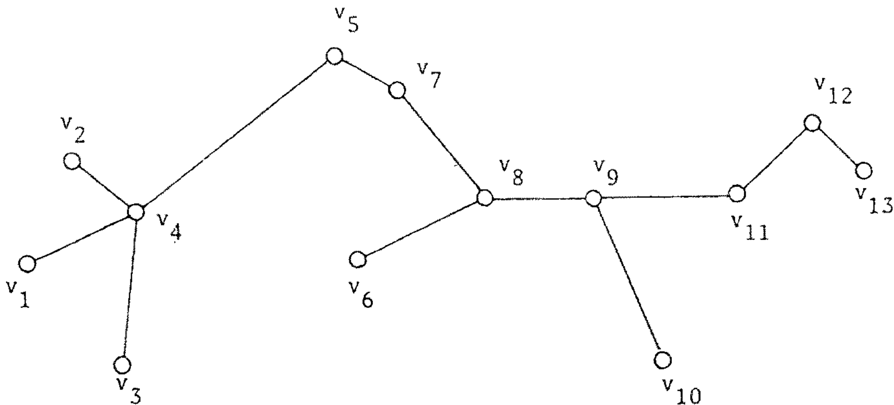

Fig. 2-2. The merged *MST* of 13 points.

Having described the algorithm informally, we are now ready to present it formally. It is called MINISPAN and is a recursive algorithm whose input is a set V of N points in the plane and whose output is a minimal spanning tree for V. Without loss of generality, we assume N even.

Algorithm MINISPAN.

Input: A set V of N points in the Euclidean plane.
Output: A minimal spanning tree of V.
Step 1: Choose a cut line L perpendicular to the X-axis such that $N/2$ points of V have X-values less than L and $N/2$ points have X-values greater than L. These two point sets are denoted V_L and V_R respectively.
Step 2: Recursively find the minimal spanning trees of V_L and V_R.
Step 3: Use the algorithm proposed by Lee and Schachter [8] to find a candidate edge set which is a subset of the Delaunay edges of V. Add the edges in the candidate edge set into $MST(V_L)$ and $MST(V_R)$ one by one from bottom to top. If a cycle is formed, delete the longest edge in the cycle.

Some observations about our algorithm are now in order. First of all, our algorithm is easy to be comprehended and hand simulated. At the very beginning the algorithm merely connects two points into an edge. Later, Delaunay edges are found and added one by one. Since Delaunay edges are quite natural in concept, they can be found visually. Whenever a cycle is found, the longest link, which is easy of find, is deleted.

Another important property of our algorithm is that unnecessary distance calculations are avoided. Now we shall prove the correctness of MINISPAN using the following lemma proved in [13].

LEMMA 1.

T is a minimal spanning tree if and only if for each non-tree edge (v, u), the length of (v, u) is at least as long as the length of any edge on the unique cycle in T formed by joining v and u.

Applying this lemma, we can prove the following:

THEOREM 1.

Algorithm MINISPAN produces a minimal spanning tree of V.

PROOF: Since we always delete one edge when a cycle is formed during the merging process, the resulting graph T is a spanning tree of V. In the sequel we shall show that the spanning tree is also minimal.

Suppose that T is not a minimal spanning tree. Then there exists an edge $(v, u) \notin T$ such that the length of (v, u) is shorter than some edge on the cycle by joining v and u.

The two vertices of edge (v, u) must belong to one of the following two classes:

Class 1: $u,\ v \in V_L$ or V_R.

Class 2: $v \in V_L$, $u \in V_R$ (or $v \in V_R$, $u \in V_L$).

If (v, u) belongs to Class 1, then either $(v, u) \in$ {edges deleted during the merging steps} or $(v, u) \in \{E_L - MST(V_L)\}$ where E_L is the edge set of V_L. But the edges deleted during the merging steps are the longest ones in each cycle formed in the merging steps of MINISPAN. Therefore $(v, u) \notin$ {deleted edges}.

Assume that $(v, u) \in \{E_L - MST(V_L)\}$. Then adding (v, u) generates two kinds of cycles, one consisting of edges only in $MST(V_L)$ and the other including some edges in V_R. For the former case, since $MST(V_L)$ is a minimal spanning tree of V_L, according to Lemma 1, (v, u) must be the longest edge in the cycle. The situation of the latter case can be described in Fig. 2-3. In this case, $e_1, e_2, \ldots, e_k \in E_L$ and $e'_1, e'_2, \ldots, e'_j \in E_R$.

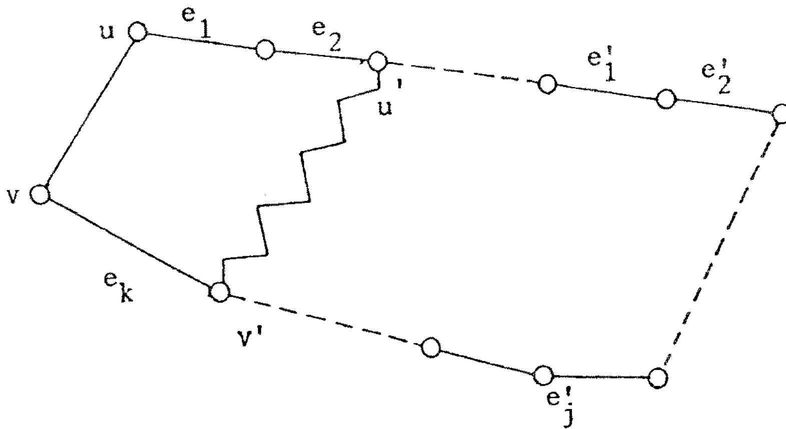

Fig. 2-3. The situation that a cycle contains edges from both V_L and V_R.

Since $MST(V_L)$ is connected, there must exist a deleted edge (u', v') which is an edge of a cycle consisting of only edges of E_L. Since $(u', v') \in MST(V_L)$, according to Lemma 1, the length of (u, v) is greater than or equal to that of (u', v). But (u', v') is deleted only if (u', v') is the longest in the cycle formed during the merging steps. Therefore, (v, u) is the longest edge in the cycle consisting of (v, u), $e_1, e_2, \ldots, e'_1, e'_2, \ldots, e'_j, \ldots, e_k$.

Assume $(v, u) \in$ class 2. Since the longest edge in any cycle is always deleted during step 3 of MINISPAN, $(v, u) \notin$ class 2. Hence (v, u) cannot exist, and according to Lemma 1, T must be a minimal spanning tree of V.

3. The complexity analysis of MINISPAN.

Let us first assume that the computation model is a random access machine (RAM) as described in [1].

Step 1 can be executed by using the linear median algorithm proposed by Blum, Floyd, Pratt, Rivest and Tarjan [5] in time $O(N)$ in the worst case. Since Step 2 solves two problems of size $N/2$, the cost of this step is $2 \cdot f(N/2)$. The timing of Step 3 needs a deeper analysis. We claim that Step 3 takes $O(N)$ time in the worst case, as indicated in the following Theorem.

THEOREM 2.
The merging step in Algorithm MINISPAN takes $O(N)$ in the worst case.

PROOF: Finding the candidate set of edges takes $O(N)$ operations [8]. Since a Delaunay triangulation contains at most $O(N)$ edges, the maximum number of candidate edges is $O(N)$. The remaining problem is whether the total number of edges traversed when cycles are formed is also $O(N)$. Since the total number of edges in a cycle formed in the merging step is $O(N)$, a brute force implementation of Step 3 will take $O(N^2)$ steps in the worst case. In the following, we shall show that edges in $MST(V_L)$ and $MST(V_R)$ do not have to be traversed more than twice.

Note that we add edges to the MST one by one and from bottom to top. If an edge (u, v) is added and a cycle C is formed as shown in Fig. 3-1, there are two possibilities:

Case 1: (u, v) is the longest in C.

Case 2: (u, v) is not the longest in C.

In case 1, (u, v) is deleted. Since the edges in C besides (u, v) form a path, the next cycle will include all these edges. However, we do not have to traverse them again because we only have to record the longest edge among e_i and compare the new candidate edge with this edge without cycle traversing. That means, we only have to traverse the edges once in this case.

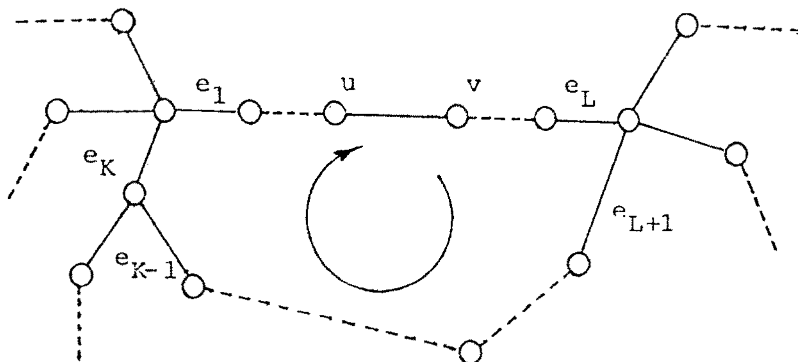

Fig. 3-1. The set of edges that we must traverse.

Case 2 is more complex. In this case, (u, v) is retained and the longest edge in the current cycle is deleted. If another cycle is formed, then the new cycle will

include some edges between e_1 and e_L, (u, v), but not any edge e_i, $L+1 \leqq i \leqq k$. Thus for such e_i, the edges are traversed only once. For (u, v) and e_i, $1 \leqq i \leqq L$, these edges are part of the edge set of next cycle and will be traversed again in the next cycle, but only once.

Thus each edge will be traversed at most twice. Hence, we conclude that the merging steps take $O(N)$ in the worst case.

Assume that the total running time of MINISPAN is $f(N)$. Based on Theorem 2, we can formulate the following recurrence relation:

$$f(N) = 2f(N/2)+O(N)$$

The solution of $f(N)$ is $O(N \log N)$. Therefore, the running time of MINISPAN is $O(N \log N)$.

Algorithm MINISPAN can readily be implemented on a tree machine, [4], as depicted in Fig. 3-2.

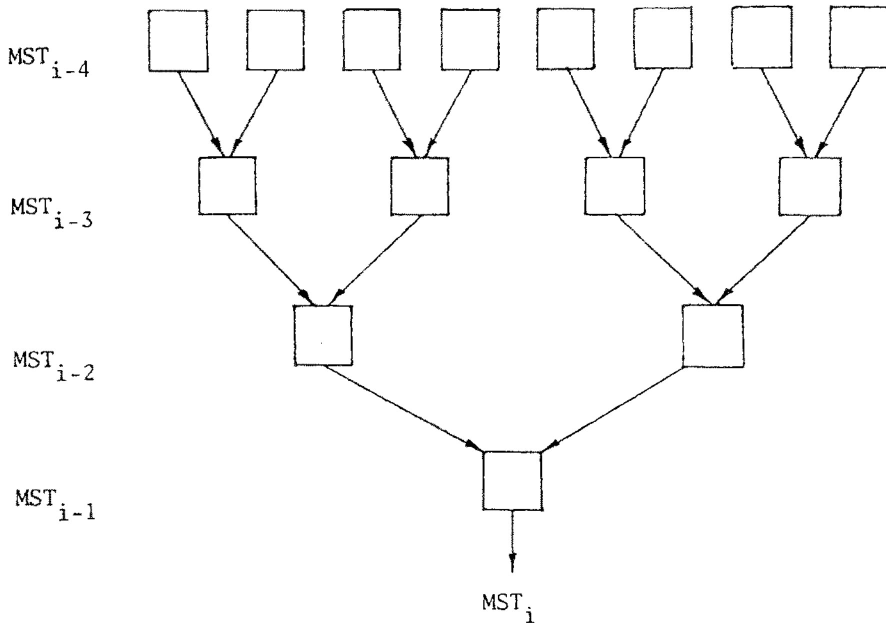

Fig. 3-2. A tree machine.

To simplify our discussion, let us denote the bottom of the tree machine to be the first level, which contains $\log N$ processors, the ancestors of the first level as the second level. In the same fashion, the ancestors of the kth level is the $(k+1)$th level. Accordingly, the root of the tree machine is in the $\lceil \log \log N \rceil$ level and each processor can be numbered as PU_{ij}, where i is the level number, and j is the sequence number of the same level (from left to right). For example, PU_{14} denotes the fourth processing unit in the first level. Assuming that we have a

drive computer, which feeds data into the bottom level of the processors, a minimal spanning tree can be constructed in parallel.

First, we feed N data points into the processors on the first level. This takes N operations. Then, each processor $PU_{1j}, j = 1, \ldots, \log N$, finds in parallel the $(N/\log N)$ data points which fall between $(N/\log N) \cdot (i-1)$ and $((N/\log N)i-1)$ (according to X-axis) points respectively. This also can be accomplished in $O(N)$ by using the linear median algorithm [5]. The third step is to find the minimal spanning trees of these $(N/\log N)$ points in the first level in parallel. This takes $O((N/\log N)\log(N/\log N)) \leqq O(N)$ by MINISPAN. The output of each PU_{1j} is a minimal spanning tree of the $N/\log N$ points. Applying the merging step of MINISPAN, we can merge these trees in the subsequent levels of processors in parallel. In the worst case, the merging step in the $(k+1)$th level needs $O((N/\log N)2^k)$ operations. The total running time needed to construct a minimal spanning tree of N vertices is

$$T_p(N) = O(N) + \sum_{k=2}^{\lceil \log\log N \rceil} (N/\log N)2^{k-1} = O(N),$$

where the first term in the equation above is the time spent in the processors of the first level.

Therefore, in the worst case, the parallel complexity of MINISPAN is $O(N)$ with $O(\log N)$ processors.

4. Concluding remarks.

In this paper, we have presented an algorithm to construct minimal spanning trees. This algorithm is based on the concept of Voronoi diagrams, and its input is a set of points in two dimensions. The algorithm is easy to comprehend and can be hand simulated, avoids unnecessary distance calculations, can be implemented as a parallel program and can construct minimal spanning trees in linear time complexity using $O(\log N)$ processors. It is different from most minimal spanning tree algorithms, except those in [3, 10] because our algorithm uses the geometrical properties of the input data. It uses many ideas in [10] suggesting to construct a minimal spanning tree by extracting it from the Voronoi diagram (actually the Delaunay triangulation). Our algorithm is more direct. It constructs a minimal spanning tree straightforwardly. It does not have to construct a Voronoi diagram because the divide-and-conquer method of constructing Voronoi diagrams can be used to construct minimal spanning trees. Our algorithm is in the same spirit as [3] since we try to avoid unnecessary distance calculations.

We like to point out here that by fully utilizing the properties of points in two dimensions, we have obtained a parallel algorithm whose complexity is $O(N)$, and

and which uses $O(\log N)$ processors. Our method of dividing elements into groups is optimal according to Tang and Lee [12].

REFERENCES

1. A. V. Aho, J. E. Hopcroft and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison Wesley, Reading, Mass. (1974).
2. J. L. Bentley, *A parallel algorithm for constructing minimum spanning trees*, Journal of Algorithms, Vol. 1, No. 1 (1980), 51–59.
3. J. L. Bentley and J. H. Friedman, *Fast algorithms for constructing minimal spanning trees in coordinate spaces*, IEEE Transactions on Computers, Vol. C-27, No. 2 (1978), 97–105.
4. J. L. Bentley and H. T. Kung, *A tree machine for searching problems*, Proceedings, IEEE 1979 International Conference on Parallel Processing (1979), pp. 257–266.
5. M. Blum, R. W. Floyd, V. R. Pratt, R. L. Rivest and R. E. Tarjan, *Time bound for selection*, Journal of Computer and System Sciences, Vol. 7, No. 4 (1972), pp. 724–742.
6. B. Delaunay, *Sur la sphere vide*, Bull. Acad. Sci. USSR (VII), Classe Sci. Mat. Nat. (1934), pp. 793–800.
7. J. Katajainen, *On the worst case of a minimal spanning tree algorithm for Euclidean space*, BIT Vol. 23 (1983), pp. 2–8.
8. D. T. Lee and B. J. Schachter, *Two algorithms for constructing Delaunay triangulations*, International Journal of Computer and Information Sciences, Vol. 9, No. 3 (1980), pp. 219–242.
9. O. Nevalainen, J. Ernvall and J. Katajainen, *Finding minimal spanning trees in a Euclidean coordinate space*, BIT, Vol. 21 (1981), pp. 46–54.
10. M. I. Shamos, *Computational Geometry*, Ph.D. Thesis, Yale University (1978).
11. M. I. Shamos and D. Hoey, *Closest point problems*, 16th Annual IEEE Symp. on Foundations of Computer Science (1975), pp. 151–162.
12. C. Y. Tang and R. C. T. Lee, *Optimal speeding up of parallel algorithms based upon divide-and-conquer strategy*, Information Sciences, Vol. 32, No. 3 (1984), pp. 173–186.
13. R. E. Tarjan, *Sensitivity analysis of minimal spanning trees and shortest path trees*, Information Processing Letters, Vol. 14, No. 1 (1982), pp. 30–33.
14. G. Voronoi, *Nouvelles applications des paramètres continus a la theorie des formes quadratiques*, Deuxieme Memoire: Recherches sur les parallelloèdres. Deuxieme angew. Math. Vol. 134 (1908), pp. 198–287.
15. A. C. Yao, *On constructing minimal spanning trees in k-dimensional space and related problems*, SIAM Journal on Computing, Vol. 11, No. 4 (1982), pp. 721–736.