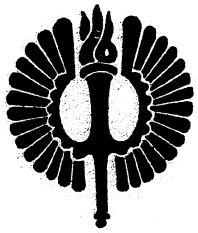

A MINIMAL SPANNING TREE ALGORITHM FOR A POINT SET IN EUCLIDEAN SPACE

by

J.Ernvall, J.Katajainen and O.Nevalainen
University of Turku
Department of mathematical sciences
Computer science
SF-20500 Finland

1.2.1980
Report A28

SARJAT:
A: Raportit
B: Käsikirjoitukset
C: Luentomonisteet
D: Käsikirjat

SERIES:
A: Reports
B: Manuscripts
C: Lecture Notes
D: Manuals

TURUN YLIOPISTO
MATEMAATTISTEN TIETEIDEN LAITOS
TIETOJENKÄSITTELYOPPI
20500 TURKU 50

UNIVERSITY OF TURKU
DEPARTMENT OF MATHEMATICAL SCIENCES
COMPUTER SCIENCE
SF-20500 TURKU 50, FINLAND

A MINIMAL SPANNING TREE ALGORITHM FOR A POINT SET IN EUCLIDEAN SPACE

by

J.Ernvall, J.Katajainen and O.Nevalainen
University of Turku
Department of mathematical sciences
Computer science
SF-20500 Finland

1.2.1980
Report A28

ABSTRACT

An algorithm for construction of a minimal spanning tree of a point set in k-dimensional Euclidean space is considered. Multiple subtrees, called point fragments, are formed in it. The minimal spanning tree is found by repeating a step in which the fragment of the minimal size is merged with the fragment which contains the point nearest to the first named fragment. Like in the algorithm of J.L.Bentley and J.H.Friedman (IEEE Trans. on Comp., p. 97, 1978) k-d tree structure and a number of priority queues are used for selecting the fragments to be connected by a new edge. At least for low dimensionalities the algorithm of this paper was fast when solving minimal spanning tree problems of normally distributed random points.

1. INTRODUCTION

Given a connected, undirected graph G=(V,E), where V is the set of nodes and E ($\subseteq$VxV) is the set of edges, and a function c tha provides each edge e ($\in$E) with a cost c(e). A spanning tree of G is a subtree of E such that there is a unique path between each two nodes of V. Our problem is to find a spanning tree T for which $\Sigma_{e\in T} c(e)$ is minimal. Such T is called a minimal spanning tree (MST) of G.

In the present paper we consider a special MST-problem, where the nodes of G are points in a k-dimensional space, edges are the straight line segments between the points and cost functio c is defined as the Euclidean distance between the endpoints of an edge. Further we assume that each point is connected with an edge to each other point of V. Our graph is thus complete containing n(n-1)/2 edges when the number of points is n.

The foregoing MST-problem can be solved by general MST-algorithms making no advantage of the nature of the graph. Such algorithms are given e.g. by R.C.Prim /Pri/, J.B.Kruskal /Kru/, A.C.Yao /Yao/ and P.Cheriton&R.E.Tarjan /CTa/. Upper limits of the time complexity of these algorithms are correspondingly

(1) $O(n^2)$, $O(m \log m)$, $O(m \log\log n)$, $O(m \log^* n)$

where n is the number of nodes and m the number of edges. Cheriton and Tarjan /CTa/ have also derived a lower limit $\Omega(m \log\log m)$ for the time complexity of a general MST-algorithm. Here $\Omega(f(m))$ denotes a function that for a positive value d exceeds d f(m) for infinite many values of m. Because in our problem m is n(n-1)/2, the computation time is due (1) long if we use the general techniques without any modifications.

If the points are in the plane (k=2) one can apply Voronoi-diagrams, as shown by M.I.Shamos and D.Hoey /SHo/. An $O(n \log n)$-algorithm can then be written. It is also shown that $O(n \log n)$ is the lower limit for the time complexity of MST-problem in 2- and k-dimensional (k>?) space /BSh/. (Algorithms using Voronoi-diagrams are outlined by Dewdney /Dew/.) F.K.Hwang /Hwa/ shows that Voronoi-diagrams can also efficiently be used for the rectilinear distance, giving the sum of absolute coordinate differences.

The progress in constructing efficient algorithms for finding the nearest neighbours /MOt/ renders possible the development of efficient methods for solving the MST-problem in k-dimensional coordinate space. For a general k the nearest neighbour problem (like many other problems in field of computational geometry) can efficiently be solved by a data structure called k-d tree. This structure was introduced by J.L.Bentley /Be1/. The nearest neighbour of an arbitrary new point can be determined in expected O(log n) time /FBF/ with this structure. The technique however requires an O(kn log n) time for preprocessing and does not support dynamic changes in the set of points to be considered.

By applying the k-d-structure in R.C.Prim's /Pri/ general MST-algorithm, J.L.Bentley and J.H.Friedman /BFr/ have developed two efficient MST-algorithms and an approximative algorithm with an observed average run time proportional to n log n. Here the spanning tree is stepwise constructed by joining subtrees, called point fragments, so that minimality and the tree property are preserved for each joined bigger subtree. This is achieved by adding to a fragment that edge which has the minimal cost and has the other end point outside the current fragment. Bentley and Friedman report particular promising results for the so-called multifragment algorithm. In it several fragments are created by starting each time with a fragment containing only a single point in low density area of the point set. The fragments found in this manner are finally connected by minimal edges.

The MST-algorithms of Bentley and Friedman use also an other advanced data-structure:a priority queue /AHU/ is used for finding the node which will have the nearest neighbour of the whole fragment. The nearest neighbour of the fragment will next be added to the current fragment and the corresponding edge will be added to the subtree. Priority queue stores for each node of a fragment the distance to the nearest neighbour. Because nodes are added to the fragment, some of the distances (priorities) become unreal in the course of the process, i.e. the nearest neighbour of a point is not outside the fragment. When such a priority is met it should be discarded, a new real distance should be determined by k-d tree and inserted into the priority queue. Thus when searching the nearest neighbour of a fragment one must possibly delete a number of unreal priorities at the front of the priority queue and insert new real priorities. In the multifragment-algorithm of

Bentley and Friedman a new fragment is introduced if more than m_0 (a given constant) unreal priorities have been removed from the front of the current fragment queue when searching for a new fragment node to be added. In the worst case one must remove all the priorities in a fragment queue before finding the first real priority. The idea of starting each fragment at low density point is supposed to reduce the mean number $\bar{m}$ of removal operations.

The MST-algorithm given in the present paper is a variation of the multifragment algorithm described above. The difference lies in the construction of the fragments; here the minimal size selection rule introduced by Cheriton and Tarjan /CTa/ is used. Now each fragment initially consists of a single node. After this at each step a new edge is added to the fragment of minimal size. This process continues until there is only one fragment left, which is in fact the wanted MST.

The proposed MST-algorithm is given in more details in Section 2. Theoretical and experimental estimates of the execution time are discussed in Sections 3 and 4, respectively.

2. MULTIFRAGMENT ALGORITHM

The following algorithm determines an MST for a point set V of k-dimensional space. The number of the points in V is n. A priority queue (called shortly the fragment queue) of minimal distances to nonfragment nodes is maintained for each fragment. Each fragment node has a notation which tells the real minimal distance, called real priority, or a lower limit of the real minimal distance, called unreal priority. The number of the fragment node and the number of the node with minimal distance are also given. Finally a priority queue of the fragment sizes is maintained to aid the selection of a fragment of minimal size.

Main steps of the algorithm are:

1. Build a k-d tree for the point set.
2. Form n single point fragments and the corresponding fragment queues. (Initially each fragment consists of a single node with no edges and each fragment queue consists of a single notation giving the distance between the point and its nearest neighbour along with the numbers of the both nodes. To avoid unnecessary distance calculation we let initially the nearest neighbour of each node be the node itself and thus give the unreal zero priorities.
3. Form a priority queue of the fragment sizes. (Initially n notations with priority 1. The fragment number is also stored in the queue.)
4. Loop until the number of fragments is one (then the MST is ready)
   do select (by using the priority queue of the fragment sizes) a fragment with minimal size;
   loop until the highest priority in the priority queue of the current minimal fragment is real do
   X ← top node in the queue;
   Y ← nearest nonfragment neighbour of X;
   Link X to Y;
   Delete the unreal priority of X and reinsert its real priority into the fragment queue;
   Repeat;
   X ← top node of the fragment queue;
   Y ← node linked to X;
   Merge the fragment queue of X with the fragment queue of Y;

```
        Insert edge (X,Y) in the fragment of Y;
        Merge the fragment of X into the fragment of Y;
        Update the fragment size priority queue by removing
          old size notations of X and Y and by reinserting a
          notation to reflex the new size of Y's fragment;
        Repeat;
    End;
```

Operation of the above MST-algorithm is based on the following

Theorem /Chr, p. 135/. Let n be the number of nodes. Start with trees $T_1, T_2, \ldots, T_n$, where each T_i consists of a single node i. Select an arbitrary tree T and link it to the tree T' for which

$$d = \min\{d(X,Y) \mid X \in T,\ Y \in T'\}$$

is minimal and $T \neq T'$. Repeat the above with the resulting trees until there is only one tree left. This tree is an MST of the whole graph.

Cheriton and Tarjan /CTa/ give a simple technique for the management of the priority queue[1] of fragment sizes; a (two-way linked) linear list is formed of the fragments containing i nodes. To facilitate the priority queue operations a vector A is used as an index of the size lists by letting the component A(i) point to the front of the list of the fragments of the size i. This allows the selection of the next fragment and the updating of the fragment sizes in O(1) time. Thus O(n) time suffices in the whole algorithm for management of fragment sizes.

Several techniques are available to implement the fragment queues /AHU/. We use in the following the so-called 2-3 trees which support the selection of the node with the greatest priority

1) Essentially the data structure needed is more general than a common priority queue. One must also be able to delete nodes from the middle of the queue. On the other hand, the values of the nodes in the queue are known to be integers from the range [1,n].

(i.e. that with minimal distance to the nearest neighbour) and the updating of an unreal priority in $O(\log n)$ time. The MST-algorithm contains also a step where two priority queues are merged. In the case of two 2-3 trees this is a simple operation and can be done in an $O(\log n)$ time.

3. COMPUTATION TIME

Four main operations can be separated in the MST-algorithm of the previous section:

1. Construction of the optimized k-d tree. This demands a computation time proportional to $kn \log n$ /FBF/. A linear time median search algorithm /AHU/ is applied in this part of the program.

2. Selection of a fragment of minimal size and updating of the priority queue of fragment sizes. As noted above, this can be done in total in a time proportional to n.

3. Determining the n-1 edges by which the fragments will be connected. Let n_i be the size of the smallest fragment and m_i the number of nearest neighbour calculations when (at the i^{th} iteration) selecting the i^{th} edge. Thus we need m_i+1 times in the fragment queue to search for the node with the smallest priority and m_i times to delete a queue element with unreal priority and reinsert it with a new real priority. With 2-3 trees this can be done in a time proportional to $(m_i+1)\log n_i$. Further, let $T_n(n_i)$ be the time of a nearest neighbour search for a node of an n_i-node fragment. In the case of optimized k-d trees the expected value of $T_n(1)$ is for large values of n proportional to $\log n$ /FBF/. However for $n_i>1$, $T_n(n_i)$ clearly depends on the configuration of the points in the fragment and on the point for which we are searching the nearest neighbour. A mathematical analysis of $T_n(n_i)$ is not known to the authors[2].

---

[2] The above problem greatly resembles the problem of finding the s^{th} closest point, analyzed in /FBF/. In the worst case we have to search for the $s=n_i+1$ closest point. According to Bentley /Be2/ the expected time to find the s^{th} closest point is proportional to $\log n + s$ in any fixed dimension. In a random case s is however most probably smaller than n_i+1 and we face a general intersection query /MOt/ which is hard to analyse.

Then altogether the n-1 edges can be chosen in a time

$$(2) \qquad H = c \sum_{i=1}^{n-1} (m_i+1)\log n_i + \sum_{i=1}^{n-1} m_i T_n(n_i).$$

4. Merging of the fragments. Each time a new edge is found, two fragments are merged. First we have to merge the fragment queues. Because n-1 fragments are merged the upper bound of the total computation time of this operation is for 2-3 trees proportional to (n-1)log n. Second we have to update the fragment numbers of the nodes in the smaller fragment: there are n_i numbers at i^{th} iteratio Fragment numbers are needed when determining whether a nearest neighbour is a fragment node or not. A list of nodes is maintained for each fragment to facilitate the updating of fragment numbers. Joining of two lists can be done in constant time. In addition, updating of fragment numbers demands in the algorithm by Theorem 2 at most a time proportional to n/2 log n. Third the edges of the fragments are given by father links. Their management demands in total at most a time proportional to n log n.

We see from the above considerations that for large values of n the only components of the processing time increasing possibly more rapidly than n log n are the first two components of (2). It is risky to draw any further general conclusions on the time complexity of the algorithm. The following notes, however, clarify the significance of the parameters.

a) Because $m_i \leq n_i$, we have $\Sigma m_i \log n_i \leq n/2(\log n)^2$. Let $\bar{m} = \Sigma m_i/(n-1)$ Then $\Sigma m_i \log n_i \leq (n-1)\bar{m} \log n$. Thus if $\bar{m}$ is small the first term of (2) does perhaps not dominate in the processing time of the whole algorithm.

b) If $T_n(1)$ is a good approximation for $T_n(n_i)$, the expected value of the second term of (2) is proportional to $(n-1)\bar{m} \log n$. Then for a small $\bar{m}$-value we can expect a good computation time also when n is large.

Next we give a theorem that states the upper limit for the number of nodes to be merged when joining the fragments. This theorem was already applied above.

Theorem 2. In the multifragment algorithm the number of nodes to be merged into the bigger fragments is bounded by

$$(1) \qquad n-1 \leq N(n) \leq (n/2)\log n.$$

Proof. The former part of (1) is obvious. Let us consider the following statement which is analogous to the latter part of the statement of the theorem: A set of n points is divided into two parts and the number of points in the smaller part is counted. Thereafter both parts are handled by the same technique (by dividing again into two parts and counting the numbers of the points in the smaller parts) etc. Finally when there are left only sets with a single node, the total number N(n) of the counted points in the smaller parts is summed up. Then this sum is less than or equal to $(n/2)\log n$.

The modified statement can be proved inductively. First, the statement is true for n=1 (then $(n/2)\log_2 n = 0$). Let us suppose that the statement is true for $n<n_0$, $n_0 \geq 2$, and prove the statement for n_0. For this we divide the set of n_0 points into two parts, the smaller of which is supposed to contain m points. Now it holds that $m \leq n/2$. Let us consider a function

$$f(n_0,m) = m+(m/2)\log_2 m+(1/2)(n_0-m)\log_2(n_0-m).$$

For a fixed n_0 and $m \in [0,n_0/2]$, $f(n_0,m)$ has its minimum at point $m=n_0/5$. Thus the maximum is attained at the other end of the interval. Because $f(0) \leq f(n_0/2)$ we have by induction

$$N(n) \leq n/2+(n/4)\log_2(n/2)+(n/4)\log_2(n/2)$$

$$= (n/2)\log_2 n.$$

This completes the proof of the theorem.

4. EXPERIMENTS

A set of experiments with the MST-algorithm of Section 2 was performed. As a function of the size n of the graph we determined

1) Y_1, the total number of nodes in the k-d tree visited during the construction of the MST (Y_1 tells the number of distance calculations and branch node visits) and
2) Y_2, the total number of times the nearest neigbour was determined

The number of points in each leaf of the k-d tree was one. This number is in the following called the bucket factor of the k-d tree.

Experimental values of Y_1/n are shown in Fig. 1 for spherically symmetric, normally distributed points in a k-dimensional coordinate space with the Euclidean distance measure. All results plotted here and in the next figures are obtained as means of 15 repetitions. At each repetition a different set of n points was drawn. In Fig. 1 we have also plotted two graphs of the form $c \cdot \log^2 n$ passing through the observed Y_1/n values for n=64. In the range $n \in [64;1024]$ Y_1/n values are clearly below the $c \cdot \log^2 n$ curve for k=2. For k=5 the increase of Y_1/n is still faster.

Fig. 2 shows that Y_2 depends linearly on n. For a fixed number of nodes the increase of the dimensionality causes the reduction of the nearest neighbour calculations. For k=2 and k=5 we have to determine the nearest neighbour in total about 2.0 and 1.5 times the number of nodes in the graph.

The observed running time as a function of the number of nodes is given in Fig. 3. For the purpose of the comparison we have also plotted the observed running time when solving the same problem with Dijkstra's general MST-algorithm /Whi/. The bucket factor of the k-d tree was now 8. In our experiments this bucket factor gave the best running time for a DEC-10 computer with a KA10-processor when using FORTRAN-IV without optimizing option.

A more detailed comparison of some MST-algorithms is made in Table 1. Here we list for the dimensionalities k=2,3,...,6 as a result of simulation experiments those n-values, for which the graphs of the running time cross the corresponding graphs of Dijkstra's algorithm. For n-values greater than the crossover points Dijkstra's algorithm takes a longer running time than that of the algorithms using a k-d tree. The given n-values for our algorithm are only approximative in nature and they greatly

depend on the details of the implementation. (Like in Fig. 3 the running time was determined only for $n = 2^6, 2^7, \ldots, 2^{10}$. Between these points an estimate of the form $c \cdot n^2$ was used for determining the running time of Dijkstra's algorithm and a linear approximation was used for determining the running time of our MST-algorithm.) The Table also shows the corresponding crossover points for the multifragment algorithm of Bentley and Friedman as taken from Table I of /BFr/. For small dimension numbers our algorithm seems to give favourable crossover points. It is however not safe to draw very strong conclusions about the efficiency of the algorithms: we do not know how good code Bentley and Friedman have used in their experiments and the behaviour of our algorithm was not tested for larger n-values.

Fig. 4 compares the observed running time T(n) of our algorithm to $n \log^2 n$. The ratio $T(n)/(n \log^2 n)$ is shown for dimensio numbers $k = 2, 3, \ldots, 7$. For $k \leq 4$ the graphs decrease slowly with increasing values of n indicating that T(n) increases more slowly than $n \log^2 n$. For values $k = 5$ to 7 the situation is no more clear. Also now the slope of the graphs becomes smaller when n grows. One possible explanation to this is, that for large k the examined n-values were not large enough to give the asymptotic running time. For instance, when n=64, there are only 4 levels in the k-d tree if we nave a bucket factor 8. The lack of extra memory prevented us to analyse the situation.

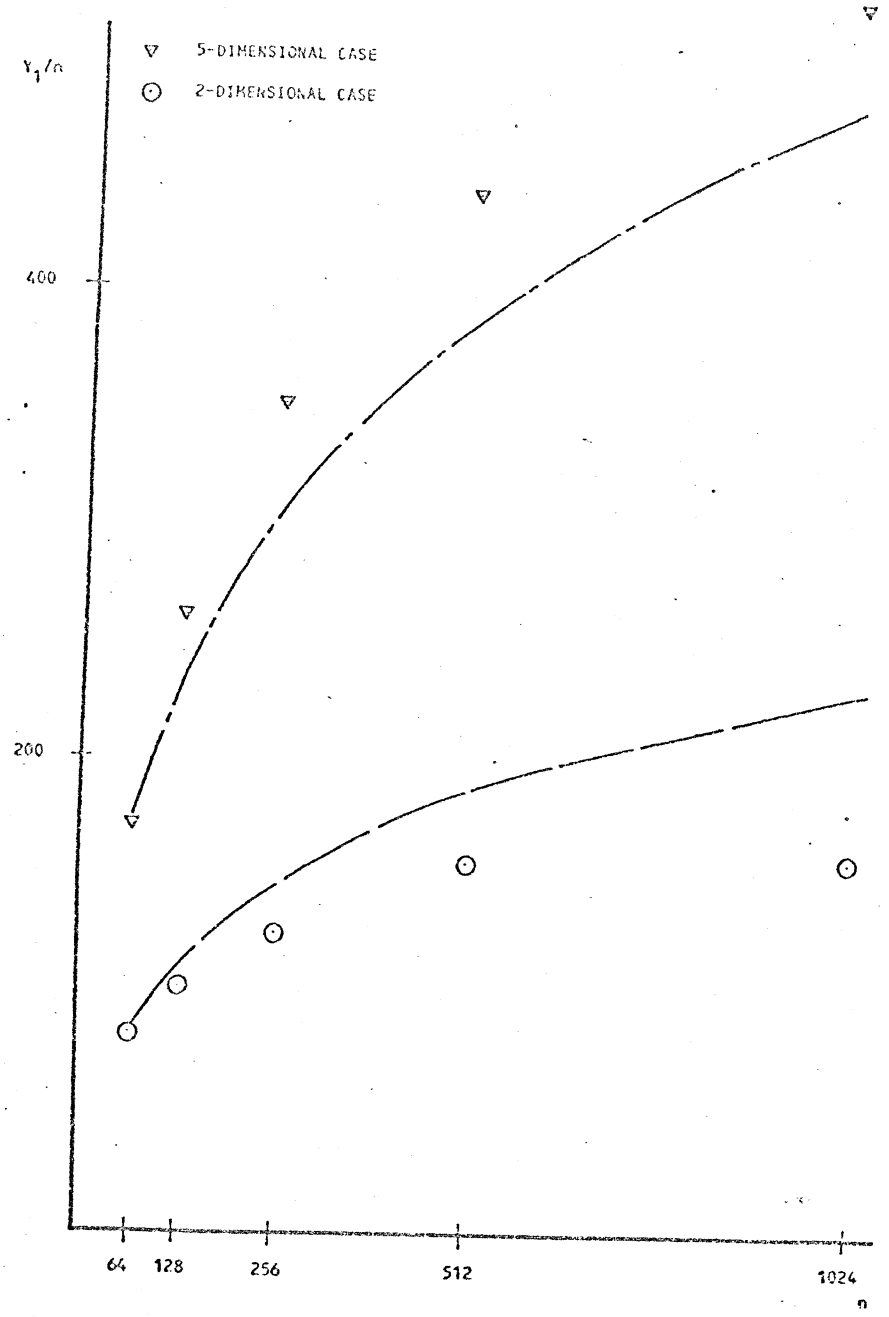

Figure 1. Y_1 (the total number of nodes in the k-d tree visited during the construction of the MST) divided by n (the number of nodes in the graph) as a function of n. Two graphs of the form $c \cdot \log^2 n$ passing through the observed Y_1/n values, for n=64, are also plotted.

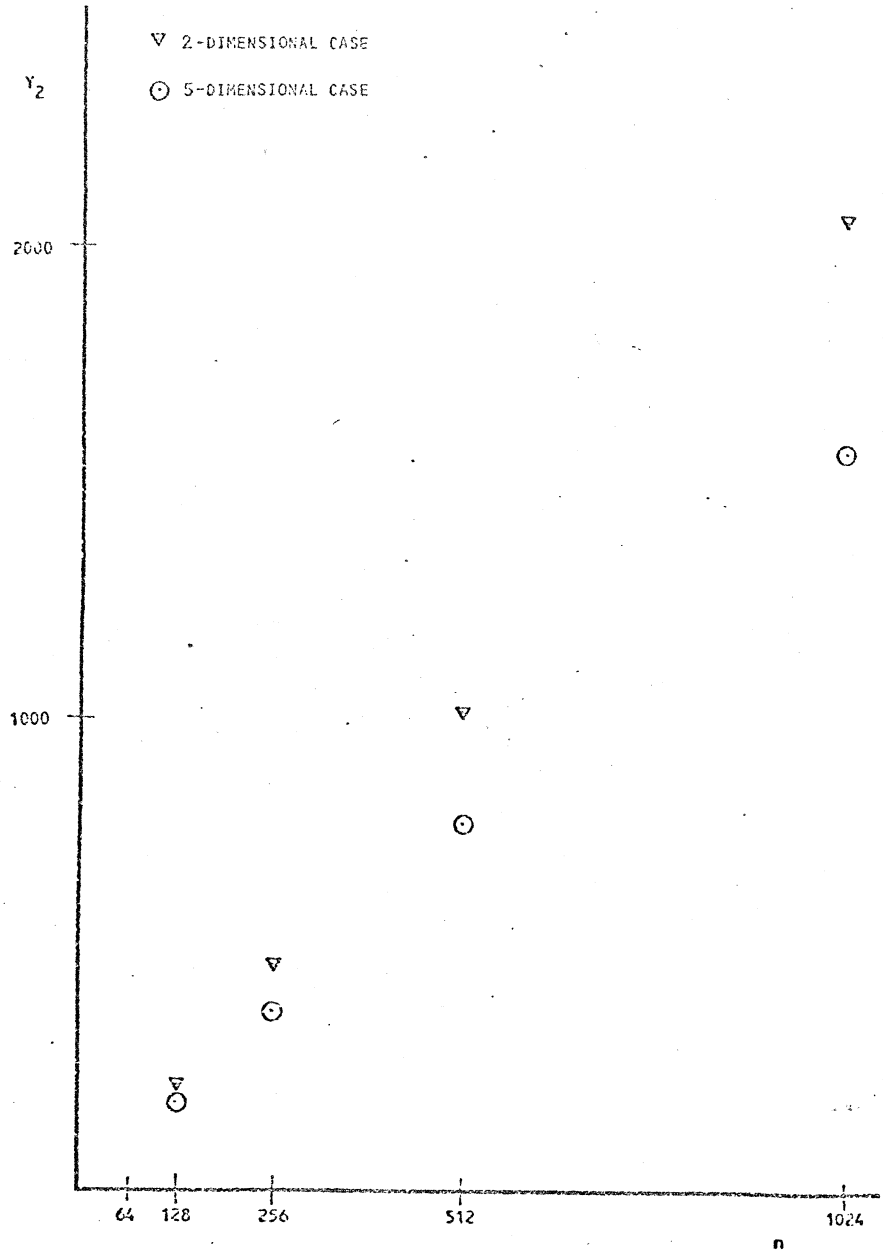

Figure 2. Y_2 (the total number of nearest neighbour calculations) as a function of n (the number of nodes in the graph).

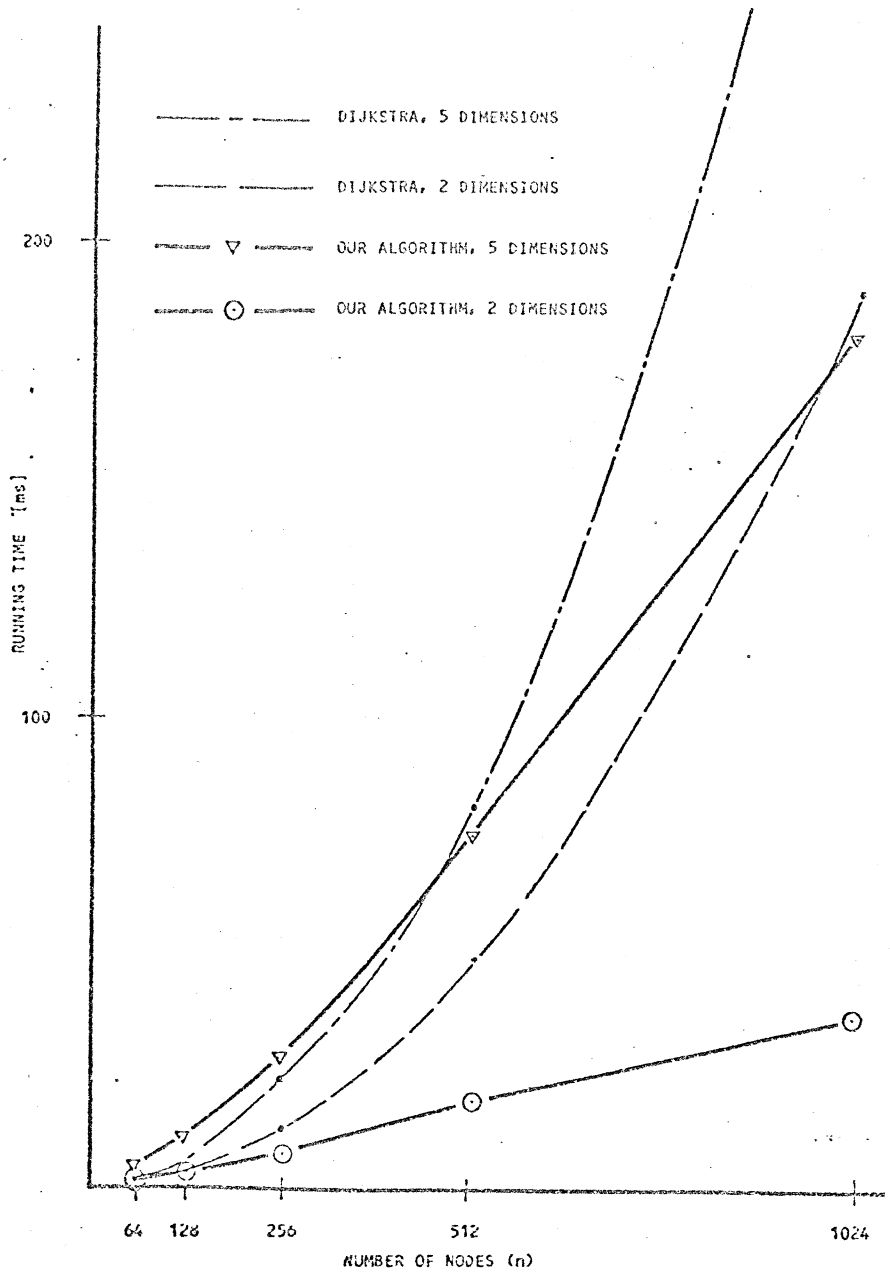

Figure 3. The total observed running time as a function of n (the number of nodes in the graph).

Table 1. Crossover points of the observed running time curves for different k-values. Multifragment algorithms are compared with the MST-algorithm of Dijkstra /Whi/. Table gives the n-value above which the corresponding multifragment algorithm gives a shorter running time than Dijkstras algorithm.

I: the algorithm of this paper,

II: the algorithm of Bentley and Friedman (from Table I of /BFr/).

k (dimensionality)	I	II
2	95	250
3	160	260
4	260	340
5	400	445
6	700	645

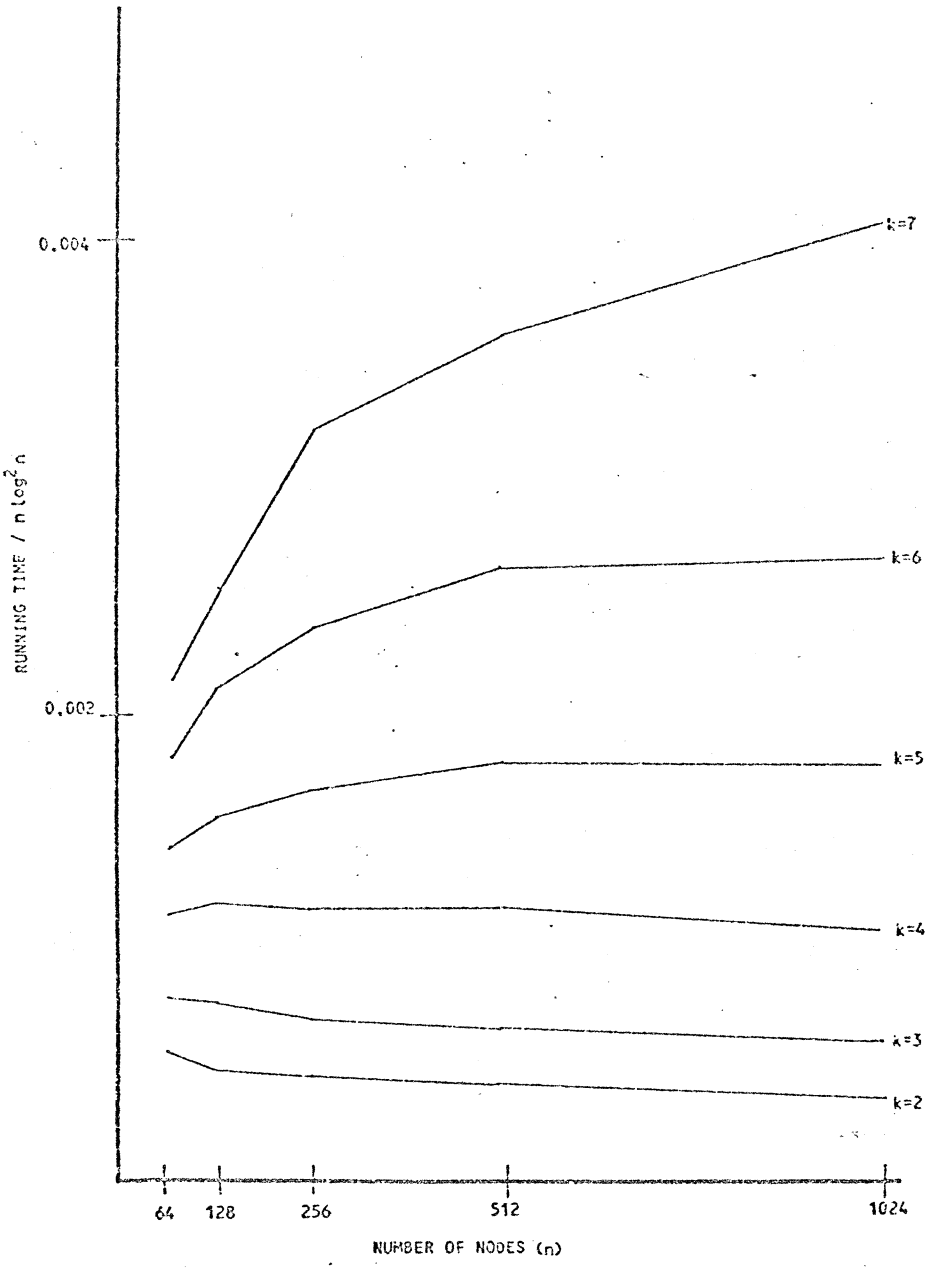

Figure 4. The total observed running time divided by $n \log_2 n$ as a function of n.

5. CONCLUSIONS

A MST-algorithm for a point set in k-dimensional space was given. The algorithm is a simplified version of the multifragment algorithm given by Bentley and Friedman /BFr/. In the original algorithm a density estimate was used for determining the starting point of a new fragment. A new fragment is introduced if more than m_0 (a predetermined fixed parameter) nearest neighbours are to be searched in order to add a new edge to the current fragment. In our simplified version of the algorithm we exclude the above two features and use as the only criterion in fragment growing the size of fragments: a new edge is always added to the fragment of minimal size. Now as before the time needed by the algorithm is mainly determined by the mean number of nearest neighbours that must be determined when searching a new edge and by the complexity of (s^{th}) nearest neighbour calculations.

Experiments with normally distributed random points indicate that our algorithm omits great deal of distance calculations and has a low running time.

Our discussion leaves open the mathematical analysis of the k-d tree structure in this particular case. Further, the metric used was Euclidean. MST-structure has also potential application in data-file compression /ENe/. Here one will rather need the Hamming-metric. With this metric one would perhaps use the search techniques described by Burkhard and Keller /BKe/ for finding the nearest neighbour.

REFERENCES

/Be1/ Jon Louis Bentley, 'Multidimensional Binary Search Trees Used for Associative Searching', Comm. ACM, Vol. 18, No.9, September 1975.

/Be2/ Jon Louis Bentley, 'Multidimensional Binary Search Trees in Database Applications', IEEE Transactions on Software Engineering, Vol. 5, No. 4, July 1979.

/BFr/ Jon Louis Bentley and Jerome H. Friedman, 'Fast Algorithms for Constructing Minimal Spanning Trees in Coordinate Spaces', IEEE Transactions on Computers, Vol.C-27, No. 2, February 1978.

/BSh/ Jon Louis Bentley and Michael Ian Shamos, 'Divide-and-conquer in multidimensional space', Proc. 8th Annual ACM Symposium on Theory of Computing, May 1976.

/BKe/ W.A. Burkhard and R.M. Keller, 'Some Approaches to Best-Match File Searching', Comm. ACM, Vol. 16, No. 4, April 1973.

/CTa/ David Cheriton and Robert Endre Tarjan, 'Finding Minimum Spanning Trees', SIAM J. Computing, Vol. 5, No. 4, December 1976.

/Chr/ Nicos Christofides, 'Graph Theory; An Algorithmic Approach' Academic Press, 1975.

/Dew/ A.K. Dewdney, 'Complexity of Nearest Neighbour Searching in Tree and Higher Dimensions', University of Western Ontario, Technical Report No.28, June 1977.

/Dij/ E.W. Dijkstra, 'A Note on Two Problems in Connection with Graphs', Numerische Mathematik, Bd.1, 269-271, 1959.

/FBF/ Jerome H. Friedman, Jon Louis Bentley and Raphael Ari Finkel, ' An Algorithm for Finding Best Matches in Logarithmic Expected Time', ACM Transactions on Mathematical Software, Vol. 3, No. 3, September 1977.

/Hwa/ F.K, Hwang, 'An O(n log n) Algorithm for Rectilinear Minimal Spanning Trees', J. of ACM, Vol. 26, No. 2, April 1979.

/Kru/ Joseph B. Kruskal, 'On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem', Proc. Amer. Math. Soc. 7, 48-50, 1956.

/MOt/ H. Maurer and Th. Ottmann, 'Manipulating sets of points - a survey', In: Graphen, Algorithmen, Datenstrukturen: Workshop 78, Hauser, München-Wien (1978).

/Pri/ R.C. Prim, 'Shortest Connection Networks and Some Generalizations', The Bell System Technical Journal, November 1957.

/Sha/ Michael Ian Shamos, 'Geometric Complexity', Proc. 7th ACM Symposium on the Theory of Computing, May 1975.

/SHo/ Michael Ian Shamos and Don Hoey, 'Closest-Point Problems', Proc 16th Annual Symposium on Foundations of Computer Science, October 1975.

/Whi/ V. Kevin M. Whitney, 'Algorithm 422 minimal spanning tree H', Collected Algorithms of CACM.

/Yao/ Andrew Chi-Chih Yao, 'An $O(|E| \log\log |V|)$ Algorithm for Finding Minimum Spanning Trees', Information Processing Letters, Vol. 4, No. 1, September 1975.