BIT 21 (1981), 46–54

# FINDING MINIMAL SPANNING TREES IN A EUCLIDEAN COORDINATE SPACE

O. NEVALAINEN, J. ERNVALL and J. KATAJAINEN

**Abstract.**

The minimal spanning tree problem of a point set in a $k$-dimensional Euclidean space is considered and a new version of the multifragment *MST*-algorithm of Bentley and Friedman is given. The minimal spanning tree is found by repeatedly joining the minimal subtree with the closest subtree. A *k-d* tree is used for choosing the connecting edges. Computation time of the algorithm depends on the configuration of the point set: for normally distributed random points the algorithm is very fast. Two extreme cases demanding $O(n \log n)$ and $O(n^2)$ operations, $n$ being the cardinality of the point set, are also given.

*Key-words*: Minimal spanning trees. Euclidean distance. Graph theory. Closest point problem, Multidimensional search trees, *k-d* tree.

## 1. Introduction

In this paper we consider the following special *minimal spanning tree* (*MST*) problem.

"Given a set of $n$ *points* in a $k$-dimensional Euclidean space, connect the points with $n-1$ straight line segments in such a way that the resulting graph is connected and the sum of the lengths of the straight line segments, called *edges*, is minimal."

One way to solve this problem is to use one of the various algorithms determining the *MST* of a connected undirected graph. In the case of a point set, the graph is complete, containing $e = n(n-1)/2$ edges and the cost of an edge is the distance between the end points. One such algorithm is that of R. C. Prim. [13] and E. W. Dijkstra [8] working in $O(n^2)$ time (abbreviated hereafter as *PDA*). For a graph with $e = \Omega(n^{1+\varepsilon})$, $\varepsilon$ being some fixed number, D. B. Johnson [10] gives an $O(e)$ version of Prim's algorithm. Here priority queues with update are used. J. K. Kruskal's greedy algorithm [11] uses a single priority queue for the selection of minimal edges and works in $O(e \log n)$ time. The edges are partitioned into subsets according to the increasing cost in A. C. Yao's $O(e \log \log n)$ algorithm [16] and delayed merging of the cost priority queues is used in the $O(e \log \log n)$ algorithm of P. Cheriton and R. E. Tarjan [5]. See also [1, p. 154 and p. 218] for two algorithms working in $O(e \log e)$ and $O(e)$ time.

Received June 4, 1980. Revised January 12, 1981.

Despite its simplicity, *PDA* is for a complete graph the most effective of the algorithms above: it works in $O(n^2)$ time also in this case, whereas for example Yao's algorithm demands $O(n^2 \log\log n)$ time.

If the points are in the plane $(k=2)$ one can apply the Voronoi-diagrams in the writing of an $O(n \log n)$ *MST*-algorithm, as shown by M. I. Shamos and D. Hoey [14]. It has also been shown that $\Omega(n \log n)$ is the lower limit for the time complexity when $k \geqq 2$ [4]. Note that $\Omega(e \log\log n)$ is the corresponding limit for a general graph. For $k \geqq 2$ A. K. Dewdney [7] outlines the design of an *MST*-algorithm using the Voronoi-diagrams.

J. L. Bentley and J. H. Friedman [3] give two fast *MST*-algorithms for the Euclidean *MST*-problem. For normally distributed random points the observed average running time is proportional to $r \log n$ for these methods. We take a closer look at the second of the *MST*-algorithms, called a multifragment algorithm by Bentley and Friedman (*BFMA*). The technique in the algorithm is first to form $n$ single point subtrees, called *point fragments.* From these we then pick one located in the lowest density area of the points. Then we may connect it to the *nearest* fragment with a minimal length edge. After this we continue by increasing the merged bigger fragment. The edges are determined by a multidimensional search tree, the *k-d* tree [2]. This structure partitions the point set into disjoint subsets of nearly the same size according to discriminator values in different dimensions and after $O(kn \log n)$ preprocessing it supports the searching for the closest point of an arbitrary point in expected $O(n \log n)$ time. Thus normally we can omit a great number of distance calculations.

Further to facilitate the choosing of the minimal edge a *priority queue* [1], that stores for each fragment point the distance to the nearest neighbour, is maintained. Because points are added to the fragment, some of the minimal distances (priorities) become unreal in the course of the process, i.e. the nearest neighbour of a point is no longer outside the fragment. Thus when searching in the priority queue for the shortest edge we must disregard the unreal priorities, recalculate them by means of the *k-d* tree and reinsert them into the queue as real priorities.

The increasing of a particular fragment is continued until either $n-1$ edges have been found or the inclusion of a new edge demands more than $t_0$ (a fixed parameter) unreal priorities to be deleted from the distance priority queue. In the first case the *MST* is completed. In the second case if single point fragments exist a new one in a low density area is again selected and the same process continues. Finally the fragments are joined with minimal edges in increasing order of the number of points in the fragments.

In the present paper we introduce a new variant of the *BFMA* described above. The difference lies in the construction of the fragments: here the minimal size selection rule of Cheriton and Tarjan [5] is used. Now each fragment initially consists of a single point. After this at each step the fragment of minimal size is joined with the closest fragment. The process continues until there is only one

fragment left, which is in fact the required *MST*. The proof that the resulting tree is really an *MST* is given by N. Christofides [6, p. 135]. In comparison to *BFMA* our algorithm (*NEKMA*) thus contains only one way of processing. The limit $t_0$ is dropped out. Further we no longer need the density estimates of the space. On the other hand, the selection of the fragment to which a new edge is added now requires somewhat more labour.

A comparison with previous algorithms indicates that our algorithm works quite well (Section 2). However, as for *BFMA* [3] we can find some special cases where the performance of *NEKMA* is either very weak or good (Section 3). A detailed version of the algorithm is given in the Appendix.

The present paper is a shortened version of two working reports by the authors [9], [12].

## 2. Computation time

Recall that in *BFMA* a new fragment is always started in a low density region of the coordinate space. The aim of this is a decreased number of distance calculations and priority queue operations. Experiments [3] with spherically symmetric normally distributed random points have given favourable results. In our algorithm the hill-climbing feature is not explicitly included and we can therefore expect weaker results for hill-like distributions.

To get an idea about the computation time of our algorithm we used the normal distribution to generate a number of point sets of different sizes ($n$ ranging from 64 to 1024) and varying dimensionalities ($k$ ranging from 2 to 7). Each time the *MST*-problem was solved 15 times for different random numbers.

Fig. 1 shows the total running time of *NEKMA* and *PDA* for $k=2$ and 5. Both algorithms were written in *FORTRAN* and run on a *DEC*-10 computer with a *KA*10-processor. The optimizing facility was not used in the compiling. It was observed that except for very small $n$-values our algorithm has a much shorter running time than the Prim-Dijkstra algorithm.

For $k \leqq 4$ the observed running time increases slower than $n (\log n)^2$. For higher dimensionalities the growth in the running time is still faster. The growth, however, smoothens for large $n$-values so that it seems probable that at least for $k \leqq 6$ the asymptotic increase is not faster than $n(\log n)^2$. The situation is not clear for $k=7$ because of the relatively small upper limit of $n$.

By using the distance priority queue and the minimal size selection rule it was our intention to decrease the number of closest point searches. If the distance priority queue were not used we might need at most $(\log_2 n)/2$ closest point searches per node [9]. However, in a case of normally distributed random points the algorithm makes only about $\bar{r}=2.0$ and 1.5 times $n$ closest point searaches for $k=2$ and 5, respectively.

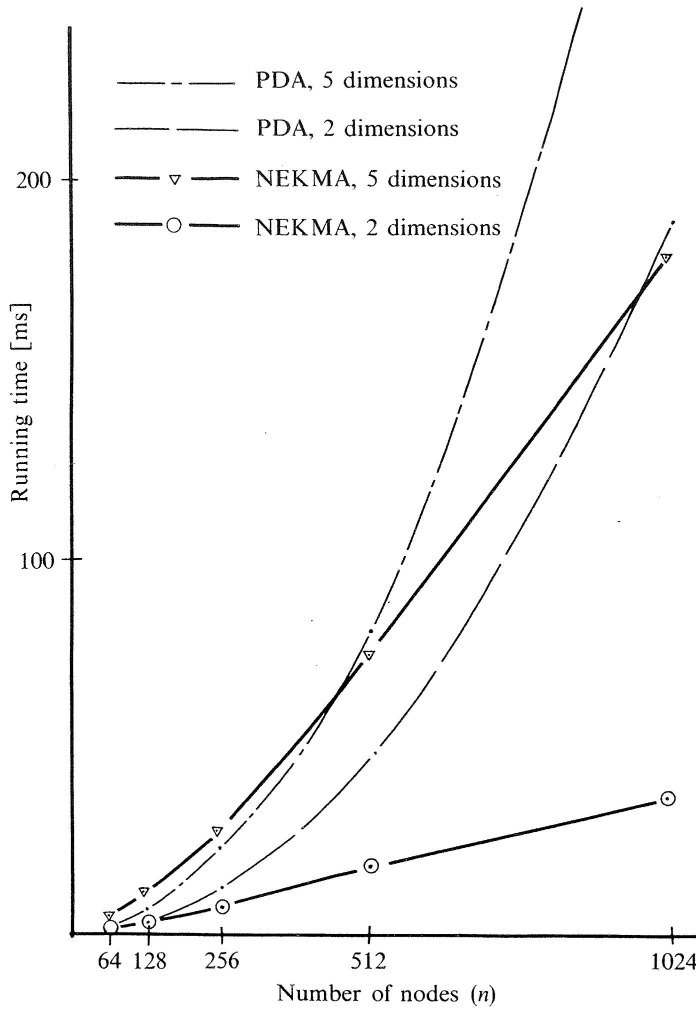


Figure 1. The observed running time as a function of $n$ (the number of nodes in the graph).
PDA: Prim-Dijkstra nearest neighbour algorithm [15],
NEKMA: Multifragment algorithm of this paper.

Table 1. *Crossover points of the observed running time curves for different dimensions* ($k$).

| $k$ | PDA vs. NEKMA | PDA vs. BFMA |
|---|---|---|
| 2 | 95 | 250 |
| 3 | 160 | 260 |
| 4 | 260 | 340 |
| 5 | 400 | 445 |
| 6 | 700 | 645 |
| 7 | >1024 | 920 |

As a result of simulation experiments, we list in Table 1 for $k=2, 3, \ldots, 7$ those $n$-values for which the curve of the running time of *NEKMA* crosses the corresponding curve of *PDA*. For larger $n$-values the time taken by the Prim-Dijkstra algorithm is longer than that of our algorithm. (The given $n$-values for *NEKMA* are only approximate and they depend on the details of the implementation. Only graph sizes $n=2^6, 2^7, \ldots, 2^{10}$ were used. For $n$-values between these an estimate of the form $cn^2$ was used for determining the running time of *PDA* and a linear approximation was used for *NEKMA*.) The table also shows the corresponding crossover points for *BFMA* as taken from [3]. For small dimension numbers our algorithm seems to give favourable crossover points. However, it is not safe to draw very firm conclusions about the relative efficiency of the two multifragment algorithms: we do not know the quality of the program used in the experiments of [3], and the behaviour of our algorithm was not tested for larger $n$-values.

We distinguish four main operations in our *MST*-algorithm:

1. Construction of the optimized $k$-$d$ tree.
2. Selection of a fragment of minimal size and updating of the priority queue of fragment sizes.
3. Determining the $n-1$ edges by which the fragments will be connected.
4. Merging the fragments.

Operations 1, 2 and 4 can be done in $O(n \log n)$ time [9]. It is hard to give an exact formula for the execution time of operation 3. The time is of the form

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

Here subscript $i$ stands for the selection of the $i$th edge, $c$ is a constant, $r_i$ the number of closest point searches when selecting the edge, $n_i$ the size of the smallest fragment, and $T_n(x)$ the time of a closest point search for a point of a fragment with $x$ points.

It is risky to draw any further general conclusions concerning the time complexity of the algorithm. The following notes, however, clarify the significance of the parameters.

a) Because $r_i \leqq n_i$, we have $\sum r_i \log n_i \leqq (n/2)(\log n)^2$. Let $\bar{r} = \sum r_i/(n-1)$. Then $\sum r_i \log n_i \leqq (n-1)\bar{r} \log n$. Thus if $\bar{r}$ is small perhaps the first term of $H$ does not dominate the running time of the algorithm.

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

## 3. Some special point sets

We next construct a point set for which the number of distance calculations is very unfavourable. For simplicity we suppose that the points lie on a straight line

and $n$ is a power of two, say $n = 2^s$ ($s \geqq 1$).

The method is to form two clusters of equal size ($n/2$) far from each other so that the algorithm first forms two separate big fragments and finally connects them. This is the case if the distance between the point clusters is so large that at the final step, when selecting the last edge, a new closest point must be searched for each point of the minimal fragment. Then, if we look back at the two big clusters, we recursively let both of them consist of two clusters of equal size ($n/4$) etc. Finally on the very first level we have $n/2$ clusters, each consisting of two points. Again the distance between the points forming a cluster is smaller than the distance between two clusters. The numbering of the points and the distances between the clusters on different levels can then be selected in such a way that at each time one has to calculate a closest point for each point in the minimal fragment. Then clearly the number of closest point searches is

$$S = n/2 + 2n/4 + 4n/8 + \ldots + 2^{s-1}n/2^s = (n/2)\log n .$$

On the other hand, by analysing the search operation in the $k$-$d$ tree we can count the total number of nodes visited in the search tree (see [12])

$$S' = 2n^2 + \frac{3n}{4}s^2 - \frac{n}{4}s = O(n^2) .$$

Second we consider a point set with decreasing density of points, as shown in Fig. 2. For this set $d(P_i, P_{i+1}) < d(P_{i+1}, P_{i+2}), i = 1, 2, \ldots, n-2$, holds. Now the number of closest point searches is $n-1$, which is minimal.

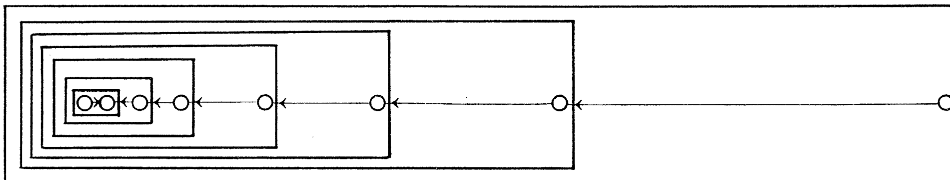

Figure 2. A point set with decreasing density. The points are indexed from left to right in the order $P_1, P_2, \ldots, P_n$.

We suppose that the point with an even index is selected for the median when constructing the $k$-$d$ tree. Then by studying the corresponding $k$-$d$ tree (Fig. 3) we can calculate the total number of nodes that must be visited in the tree

$$S' = (s+4)(n-1) + \sum_{i=1}^{s-1} n2^{-i}(3(i+1)-1) - 5 = O(n \log n) .$$

Here the first term in the sum originates from the effort to reach the actual point, to visit the neighbouring point in the three-point subtree and to return to the root of the three-point subtree. The remaining terms of $S'$ result from visiting the other longer branch.

If an odd index is selected for the median when constructing the $k-d$ tree, a somewhat smaller value of $S'$ is obtained. $S'$ is nevertheless $O(n \log n)$.

If the order of the points is reversed, i.e. $d(P_i, P_{i+1}) > d(P_{i+1}, P_{i+2})$, for $i=1, 2, \ldots, n-2$, the situation changes only a little and the number of closest point searches is low (now $n$). Note that the algorithm is sensitive to the order of the points. We can re-number the points in Fig. 2 in such a way that $O(n^2)$ leaf nodes are to be visited in the $k-d$ tree.

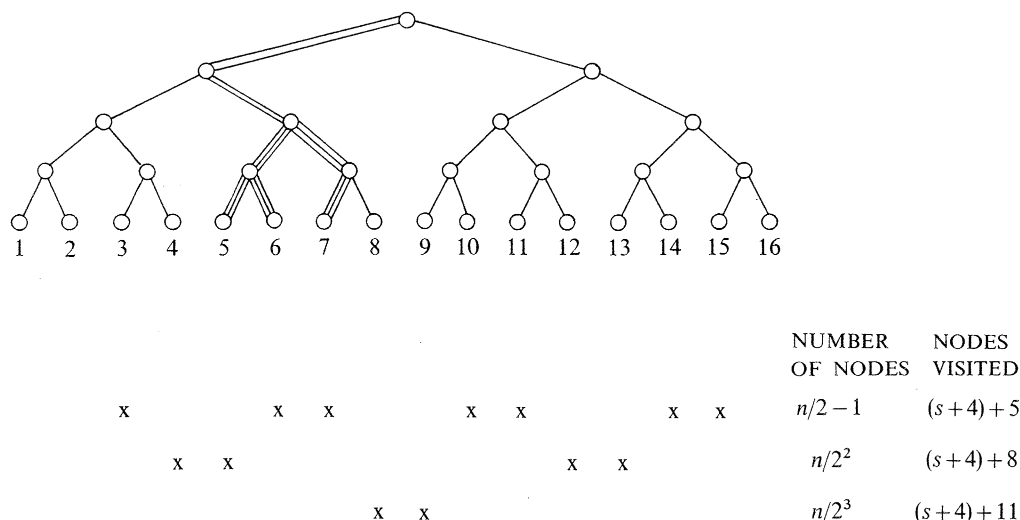


Figure 3. Counting the number of node visits in the $k-d$ tree for a point set with decreasing density. The path of the search process is shown for $P_6$ (heavy lines).

## 4. Conclusions

A different version of the multifragment *MST*-algorithm of Bentley and Friedman was given. The running time of the algorithm is mainly determined by the number of closest point searches when searching for a new edge and by the effort of searching for the ($v$th) closest point.

Experiments with normally distributed random points indicate that the algorithm omits many distance calculations and has a low running time. The observed number of closest point searches for $k=2$ was only about $2n$.

In the worst case $O(n \log n)$ closest point searches are made in the algorithm. Then the number of node visits in the $k-d$ tree is $O(n^2)$. We do not claim, however, that this is the maximum number of node visits.

The multifragment algorithm using the minimal size selection rule works very efficiently if the points happen to form a special pattern in the Euclidean space: for a point set of decreasing density the number of node visits in the $k-d$ tree is $O(n \log n)$.

## APPENDIX

The following algorithm determines a minimal spanning tree for a point set in a $k$-dimensional Euclidean space. The technique is to start with $n$ single point fragments and repeat $n-1$ times a construction step where a fragment of minimum size is selected and joined with a minimal edge to the nearest fragment.

Data structures:

- A *$k-d$ tree* of the points is used to aid the calculation of minimal distances.
- A *priority queue of the fragment sizes* helps in the selection of the fragments; an indexed linked list here works effectively, [5].
- A priority queue of minimal distances to non-fragment points is maintained for each fragment. Hereafter these are called *fragment queues.* Because fragment queues must be merged, 2-3 trees are used in the implementation.

Multifragment algorithm (*NEKMA*):

1. *Construct* an optimized $k-d$ tree for the point set.
2. *Form* $n$ single point fragments and the corresponding fragment queues. (Initially each fragment consists of a single point 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 indexes of both points. To avoid unnecessary distance calculations we initially let the nearest neighbour of each point be the point itself and thus get the unreal priorities equal to zero.)
3. *Form* a priority queue of the fragment sizes. (Initially $n$ notations with priority 1.)
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*
   $X \leftarrow$ top node in the queue;
   $Y \leftarrow$ closest nonfragment point to $X$;
   *Link* $X$ to $Y$;
   *Delete* the unreal priority of $X$ and *reinsert* its real priority into the fragment queue;
   *Repeat*:
   $X \leftarrow$ top node of the fragment queue;
   $Y \leftarrow$ 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 the old size notations of $X$ and $Y$ and by reinserting a notation to reflect the new size of $Y$'s fragment;
   *Repeat*;

## REFERENCES

1. Alfred Aho, John Hopcroft and Jeffrey Ullman, *The Design and Analysis of Computer Algorithms*, (Addison-Wesley, 1974).
2. Jon Louis Bentley, *Multidimensional binary search trees used for associative searching*, Comm. ACM, Vol. 18, No. 9, September 1975.
3. Jon Louis Bentley and Jerome H. Friedman, *Fast algorithms for constructing minimal spanning trees in coordinate spaces*, IEEE Trans. on Computers, Vol. C-27, No. 2, February 1978.
4. Jon Louis Bentley and Michael Ian Shamos, *Divide-and-conquer in multidimensional space*, Proc. 8th Ann. ACM Symp. on Theory of Computing, May 1976.
5. David Cheriton and Robert Endre Tarjan, *Finding minimal spanning trees*, SIAM J. Computing, Vol. 5, No. 4, December 1976.
6. Nicos Christofides, *Graph Theory; An Algorithmic Approach*, (Academic Press, 1975).
7. A. K. Dewdney, *Complexity of nearest neighbour searching in three and higher dimensions*, Univ. of Western Ontario, Techn. Rep. No. 28, June 1977.
8. E. W. Dijkstra, *A note on two problems in connection with graphs*, Numerische Mathematik, Bd. 1, 269-271, 1959.
9. Jarmo Ernvall, Jyrki Katajainen and Olli Nevalainen, *A minimal spanning tree algorithm for a point set in Euclidean space*, Rep. 24, Comp. Sci., Univ. of Turku, Finland, 1980.
10. D. B. Johnson, *Priority queues with update and finding minimal spanning trees*, Inf. Proc. Letters, Vol. 4, No. 1, 1975.
11. Joseph B. Kruskal, *On the shortest spanning subtree of a graph and the traveling salesman problem*, Proc. Amer. Math. Soc. 7, 48-50, 1956.
12. Olli Nevalainen and Jarmo Ernvall, *A note on a minimal spanning tree algorithm for Euclidean space*, Rep. 25, Comp. Sci., Univ. of Turku, Finland, 1980.
13. R. C. Prim, *Shortest connection networks and some generalizations*, The Bell Systems Techn. J., November 1957.
14. Michael Ian Shamos and Don Hoey, *Closest-point problems*, Proc. 16th Ann. Symp. on Found of Comp. Sci., October 1975.
15. V. Kevin and M. Whitney, *Algorithm* 422 *Minimal spanning tree H*, Collected Algorithms of CACM.
16. Andrew Chi-Chih Yao, *An* $O(|E| \log\log |V|)$ *algorithm for finding minimum spanning trees*, Inf. Proc. Letters, Vol. 4, No. 1, September 1975.

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