ON THE WORST CASE OF A MINIMAL SPANNING TREE

ALGORITHM FOR EUCLIDEAN SPACE

Jyrki Katajainen

University of Turku

Department of mathematical sciences

Computer science

SF-20500 Finland

ON THE WORST CASE OF A MINIMAL SPANNING TREE

ALGORITHM FOR EUCLIDEAN SPACE

Jyrki Katajainen
University of Turku
Department of mathematical sciences
Computer science
SF-20500 Finland

ABSTRACT

   This paper concerns the worst case running time of the
minimal spanning tree algorithm presented by Bentley and
Friedman.

   For a set of N points in k-dimensinal Euclidean space
the worst case performance of the algorithm is shown to
be $\Theta(N^2 \log N)$, for $k \geq 2$ and $\Theta(N^2)$, for $k = 1$.

Keywords: minimal spanning tree, Euclidean space, k-d tree.

ON THE WORST CASE OF A MINIMAL SPANNING TREE ALGORITHM FOR EUCLIDEAN SPACE

Jyrki Katajainen

Department of Mathematical Sciences
University of Turku, Finland

1. Indroduction

We shall study the special minimal spanning tree (MST) problem
where a set of N points in k-dimensional Euclidean space is to be
connected in such a way that the sum of the lengths of the N-1
straight line segments is minimal. The problem can be solved by
the method of Bentley and Friedman /1/. We shall discuss the
running time of this algorithm in the worst case. Nevalainen,
Ernvall and Katajainen /2/ gave a simplified version of the
algorithm. It is easy to see that the worst case of these
MST-algorithms is the same. So we shall consider only the
simplified version.

The algorithm of /2/ functions as follows. At first N single point
subtrees called fragments are formed. After this, at each step,
the fragment of minimal size is joined to the nearest fragment
with a minimal line segment. The process continues until there is
only one fragment left which is the MST. To facilitate the
choosing of the minimal distance a priority queue that stores the
distance to the nearest neighbour for each point of a fragment is
maintained for each fragment. Because points are added to the
fragment in the course of the process, for some points their
nearest neighbour is no longer outside the fragment. If this kind
of distance is met on the top of the priority queue we must search
the nearest neighbour of the point outside the fragment by means
of a k-d tree /1/ and update the queue.

The k-d tree is a binary tree in which each node represents a
subcollection of the points. In an internal node it is told how
the subcollection is partitioned into two disjointed subsets. At
each node the partitioning is accomplished by dividing the
subcollection at the median value of one of the coordinates. The
root of the tree represents the entire point set and the leaf
nodes contain the actual points. With each node a set of bounds is
associated, which jointly form a rectilinearly oriented
hyperrectangle in the k-space, within which all the points of the
particular subcollection must lie.

The nearest neighbour of a point is searched among the points
outside the particular fragment by using the k-d tree and in this
paper this is called the nearest outside neighbour searching. The
searching can be described recursively. For an internal node we
first search the subtree in which the discriminator coordinate of
the points is on the same side of the partition value as the
coordinate of the given point. If any part of the hyperrectangle
of the opposite subtree do intersect a ball centered the given

point, with radius equal to the current nearest outside neighbour
distance, then that subtree must also be searched. Generally, for
a leaf node, the distance from the given point is calculated but
in that case the point in the leaf node belongs to the same
fragment as the given point the leaf node is rejected at once.

Zolnowsky /3/ considered three different choices for the selection
of discriminator coordinates. He showed how the performance of the
k-d tree depends on the choice in context of the usual nearest
neighbour searching. We shall apply his ideas and show that the
running time of our MST-algorithm is $\Theta(N^2 \log N)$ [1] in the worst case,
for k>2, if the discriminator is the coordinate in which the
points have the greatest spread in values. In /2/ a
one-dimensional point set was given for which the running time is
$\Omega(N^2)$. We shall show that $O(N^2)$ is the worst case upper bound for
k=1.

2. The worst case study

2.1 The upper and lower bounds for k>2

In the MST-algorithm the number of the points to be merged into
the bigger fragments is at most $(N/2)\log N$ /2/. In the worst case,
for all of these points the nearest outside neighbour has to be
searched. The worst thing that could happen is that the entire k-d
tree must be searched each time. So we have a straighforward upper
bound $O(N \log N)$ for the worst case complexity of the algorithm.

Now let us give a point set in the two-dimensional space for which
the running time of the MST-algorithm reaches the above upper
bound. Let us assume the number of the points is a power of 2, say
$N=2^s$ (s>2). We choose the first N/2 points as

$$\left(s+\sum_{j=0}^{s-2}\left\lfloor\frac{i-1}{2^j}\right\rfloor,\ 0\right)\ ,\ \text{for } 1\leq i\leq N/2$$

and the next N/2-1 points as

$$\left(0,\ \frac{1}{2^{N-i}}\right),\ \text{for } N/2<i<N.$$

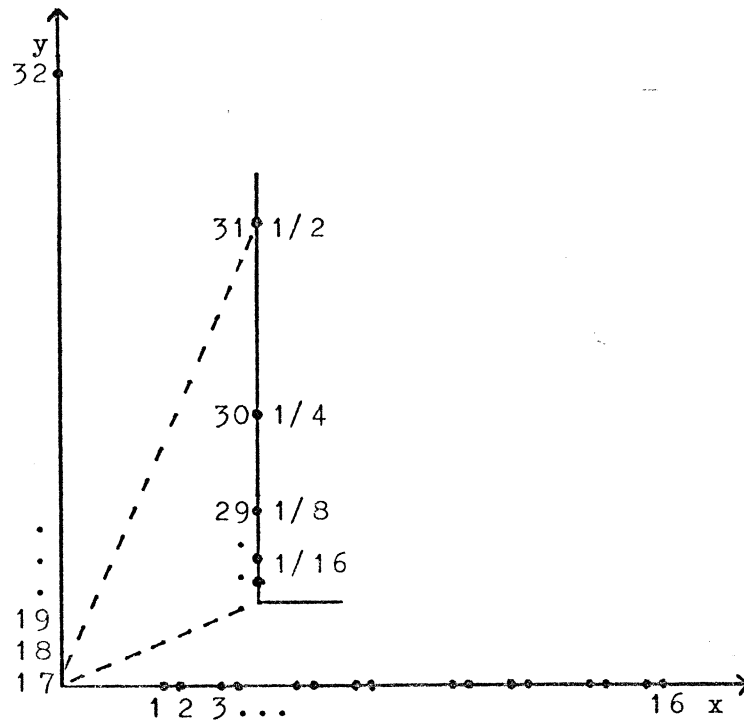Finally we choose the N th point as $(0,N)$ (See Figure 1).

Figure 1. An example of the worst case point set for n=32.

Because the point set on the y-axis is expressed with decreasing density all of the points belong to the same fragment after the fragments of size one have been considered. On the x-axis we get first $N/4$ fragments of equal size, then $N/8$ fragments of equal size etc. We know that the number of the nearest outside neighbour searches is totally $(N/4)\log(N/2)+N/2$ /2/ because every priority in a priority queue of a fragment on the x-axis must be updated.

For these points the spread in the y-coordinate is one unit greater than that in the x-coordinate. Thus the discriminator of the root of the k-d tree is the y-coordinate. The points on the y-axis and those on the x-axis are put into the different subcollection. The points on the y-axis have zero spread in the x-coordinate. So the discriminator coordinate is the y-coordinate for each node in the subtree representing the subcollection. Hence the search ball of each point on the x-axis must intersect the rectangle of each point on the y-axis.

The number of the nearest outside neighbour searches for this point set is $\Omega(N \log N)$ each demanding $\Omega(N)$ operations. Now the running time of the MST-algorithm is $\Omega(N^2 \log N)$. To sum up we know the algorithm has a time complexity that is $\Theta(N^2 \log N)$ for k>2.

2.2 The upper bound for k=1

In one-dimensional space points are on a line. The fragments consist of adjacent points on the line. In the worst case during the search of the nearest neighbour of a fragment for each point of that fragment the nearest outside neighbour must be determined. It is further possible that in connection with each search all the leaf nodes containing a point of the fragment and two successive leaf nodes must be visited in the k-d tree. Thus in the worst case, m+2 adjacent leaf nodes are visited when the size of a fragment is m.

We are going to count the number of the nodes visited in the k-d tree during the search procedure. Since a constant time is spent in each node when searching the nearest neighbour, the number of the nodes counted in this way indicates the complexity of the algorithm.

We denote the <u>closest common ancestor</u> of the fragment F (cca(F)) by a node which is the root of the smallest possible subtree containing all of the adjacent m+2 leaf nodes. Let $F'\subseteq F$ be the subset which contains the points in the leaf nodes of the left subtree of cca(F) and F" the corresponding subset for the right subtree of cca(F). The complete subtree, whose root is cca(F') is called A and the subtree whose root is cca(F") is called B. In Figure 2 the situation is described in the case where cca(F) is the root of the whole tree.
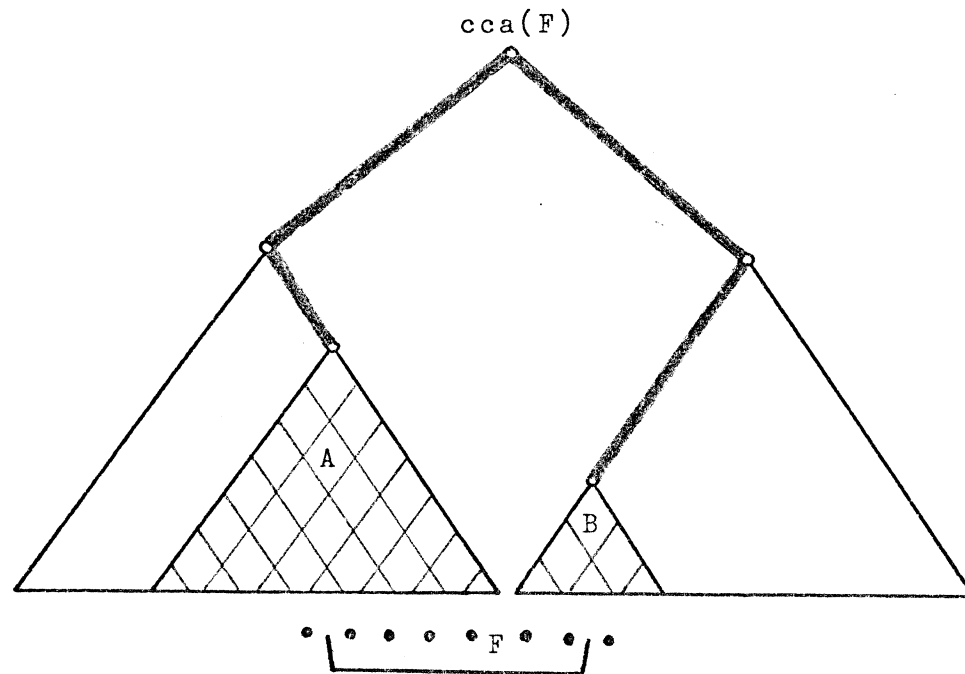
Figure 2. The nodes that must be visited when the nearest outside neighbour is searched for a point of F are on the path made dark, and in the subtrees A and B.

The nodes visited during the search are on the path from the root of the k-d tree to cca(F') and to cca(F") and in the subtrees A and B. We shall first count the number of the nodes on the path. The number of the nodes on the path is at most 2h where h is the height of the k-d tree. At most $O(N \log N)$ nearest outside neighbour searches have to be done /2/. Thus the total number of the nodes visited in order to reach both of these two nodes is at most $O(N \log^2 N)$.

When we count the number of the nodes in the subtrees A and B we get a rough upper bound for the number of the nodes actually visited in these subtrees during the nearest outside neighbour search. Now it is easy to see that the number of the nodes in these subtrees cannot be more than 4m if the size of the fragment is m. The maximal number of the searches is m for a fragment. Let T(n) represent the number of the nodes visited in these subtrees before the fragment with n points has been reached. An upper bound for the number is given by the recurrence relation

$$T(1) = 0;$$

$$T(n) \leq T(m)+T(n-m)+m*4m , \quad n>1$$

where $m<n/2$ is the size of the smaller fragment in the previous stage. Now it can be proved inductively that $T(N) \leq 2N^2$.

In the worst case during the MST-algorithm $O(N^2)$ nodes must be visited. Thus an upper bound for constructing the MST by the algorithm is $O(N^2)$ for k=1. When the lower bound result from /2/ is taken into account we get that the running time of the algorithm is $\Theta(N^2)$ in the worst case for k=1.

3. Refinements to the algorithm

Instead of using the greatest spread partition criteria in the construction of the k-d tree the so-called square criteria could give better running times in the worst case as seen in /3/. The idea is to choose the discriminator as the coordinate in which the separation of the bounds is currently greatest. However, in one-dimensional space the two k-d tree structures are the same. Thus the point set presented in /2/ is still valid in order to get $\Omega(N^2)$ performance. This point set has just the property that during the nearest outside neighbour searching in the k-d tree all the leaf nodes containing a point of that fragment must be visited. Additionally this search must be carried out for every point of the fragment.

As regards the performance of the MST-algorithm, an essential refinement is achieved for k=1 if during the search of the nearest outside neighbour an entrance to the branches whose every point belongs to the same fragment as the given point is not allowed. Bentley and Friedman called this "poisoning". Then a new label must be added to the nodes of a k-d tree. The label tells the name of the fragment to which all the points of the corresponding subtree belong. If the left subtree and the right subtree of a node contain points from different fragments the label is undefined.

The updating of the labels takes $O(N \log^2 N)$ time. For each point of a smaller fragment we must first search the leaf node of that point and then backtrack towards the root until the updating is no more possible. At most $O(N \log N)$ update operations are performed each demanding $O(\log N)$ time. After this refinement each nearest outside neighbour search takes no longer $O(2\log N)$ time because of the special form of the fragments. The number of the nearest outside neighbour searches is at most $O(N \log N)$ and so the algorithm takes $O(N \log^2 N)$ time in the worst case. It is an open question whether the method would be applied successfully even to k>1.

It should be pointed out that after this refinement the algorithm is not yet optimal, since $O(N \log N)$ time is possible by the sorting. Perhaps, it is not sensible to replace the search of the nearest neighbour of a fragment with many nearest outside neighbour searches. During the algorithm the nearest neighbour of a fragment must be searched N-1 times and in the worst case this has to be replaced with $(N/2)\log N$ nearest outside neighbour searches. It is worth studying if there exists a data structure

which supports an efficient search of the nearest neighbour of a fragment.

We have seen the priority queues became useless because it is possible to construct point sets for which the nearest outside neighbour search is needed to each point of the fragment until surely can be said which point is the nearest. When an algorithm with a good performance in the worst case is formed, priority queues need not be used this way. However, the use of the priority queues is fundamental in the algorithm of Bentley and Friedman where the fragment is grown from the area with the lowest local density of points towards the points closer each other.

Acknowledgment

References

/1/ Jon Bentley and Jerome Friedman: Fast Algorithms for Constructing Minimal Spanning Trees in Coordinate Spaces, IEEE Trans. on Computers, Vol. C-27, No. 2, 1978, pp 97-105.

/2/ Olli Nevalainen, Jarmo Ernvall and Jyrki Katajainen: Finding Minimal Spanning Trees in a Euclidean Coordinate Space, BIT 21, 1981, pp 46-54.

/3/ John Zolnowsky: Topics in Computational Geometry, Stanford University, Ph.D., 1978.