

On the Euclidean Minimum Spanning Tree Problem*

Sanguthevar Rajasekaran

Dept. of CSE, University of Connecticut

Storrs, CT 06269

Abstract

Given a weighted graph $G(V, E)$, a minimum spanning tree for G can be obtained in linear time using a randomized algorithm or nearly linear time using a deterministic algorithm. Given n points in the plane, we can construct a graph with these points as nodes and an edge between every pair of nodes. The weight on any edge is the Euclidean distance between the two points. Finding a minimum spanning tree for this graph is known as the Euclidean minimum spanning tree problem (EMSTP). The minimum spanning tree algorithms alluded to before will run in time $O(n^2)$ (or nearly $O(n^2)$) on this graph. In this note we point out that it is possible to devise simple algorithms for EMSTP in k -dimensions (for any constant k) whose expected run time is $O(n)$, under the assumption that the points are uniformly distributed in the space of interest.

*This research has been supported in part by the NSF Grants CCR9912395 and ITR0326155.

1 Introduction

Finding a minimum spanning tree of a given weighted graph $G(V, E)$ is an important graph problem. This problem has been studied extensively and numerous (sequential, randomized, and parallel) algorithms have been devised for its solution. Asymptotically optimal randomized algorithms have been developed (see e.g., [3]). These algorithms run in time $O(|V| + |E|)$ with high probability. Here the high probability is over the space of all possible outcomes for the coin flips made in the algorithm. A deterministic algorithm with a run time of $O(|E|\alpha(|E|, |V|))$ has been given in [1] where α is the inverse Ackerman's function.

The Euclidean minimum spanning tree problem (EMSTP) is defined as follows. Input are n points in the plane. Consider a graph $G(V, E)$ whose nodes are the points in the plane. There is an edge between every pair of nodes in V . The weight on any such edge is the Euclidean distance between those two points. The problem is to find a minimum spanning tree for this graph. Clearly, the algorithms mentioned above could be employed to solve this problem. In this case the run time will be $O(n^2)$ (or nearly $O(n^2)$).

Shamos and Hoey [5] have presented an $O(n \log n)$ time for this problem. Yao [6] considers the EMSTP in k -dimensions (for $k \geq 3$). In particular, he has presented an algorithm with a run time of $O(n^{2-a(k)}(\log n)^{1-a(k)})$ where $a(k) = 2^{-(k+1)}$. When $k = 3$ an improved algorithm with a run time of $O((n \log n)^{1.8})$ has also been given in [6].

In this note we point out that one could solve the EMSTP problem in k -dimensions in an expected $O(n)$ time using very simple algorithms, as long as k is a constant. Here the expectation is computed assuming that the input points are uniformly distributed in the space of interest.

We first present an algorithm that has an expected run time of $O(n)$ in the plane. Followed by this, we extend the algorithm to higher dimensions.

2 An Expected Linear Time Algorithm on the Plane

The algorithm to be presented is a variant of the Prim's algorithm [4]. Thus we summarize Prim's algorithm first. This algorithm grows the minimum spanning tree one edge at a time starting with the edge of minimum weight. At any given time the algorithm keeps only one tree (which is a subtree of the final MST). There are $n - 2$ phases in the algorithm and in each phase a new edge is added to the subtree. We denote the subtree at the beginning of phase $i, 1 \leq i \leq n - 2$, as T_i . In particular, T_1 has only one edge, namely, the minimum weight edge.

Consider an arbitrary phase i ($1 \leq i \leq n - 2$). If (u, v) is the edge that gets added in this phase, then the following are true: 1) The node u is in T_i and v is outside T_i ; 2) From among all the edges that go out of T_i , (u, v) has the least weight. Let S_i denote the set of all edges that go out of T_i (i.e., each such edge has one end point in T_i and the other outside T_i); 3) After adding the edge (u, v) into T_i , S_i gets modified to include edges from v to the neighbors of v that are outside T_i and to exclude (u, v) .

Observation 1: For Prim's algorithm to work correctly, it is not necessary for S_i to include all the edges that go out of T_i . As long as S_i includes the minimum weight edge that goes out of T_i , the algorithm works correctly (independent of what other edges are in S_i).

The idea behind our algorithm is the following. We generate a graph $G(V, E)$ where V is the set of input points and E consists of edges from every input point p to points within a certain neighborhood of p . $|E| = O(|V|)$ with high probability. (By high probability we mean a probability of $\geq (1 - n^{-\alpha})$ for any fixed $\alpha \geq 1$.) We use the algorithm of [3] (for example) to find a minimum spanning forest F of $G(V, E)$. If F has only one tree, then we output this and quit. Let $T(V', E')$ be the tree in F that has the largest number of nodes. Let $V'' = V - V'$. For every point p in V'' we include edges from p to points in a sufficiently large neighborhood of p . If these edges together with E' are enough to produce a minimum spanning tree for all the input points, we output this and quit. If not we expand

the neighborhood of points in V'' and repeat this process until we succeed. More details follow.

Let the input points be p_1, p_2, \dots, p_n . Without loss of generality assume that these points are uniformly distributed within a unit square in the plane. In particular if (x, y) is an input point, then both x and y are independent random variables uniformly distributed in the interval $(0, 1)$.

Partition the square into a $\sqrt{n} \times \sqrt{n}$ square grid with n cells the size of each cell being $\frac{1}{\sqrt{n}} \times \frac{1}{\sqrt{n}}$. We refer to the length $\frac{1}{\sqrt{n}}$ as one *unit* from hereon. Label these cells as $g_{i,j}$ for $1 \leq i, j \leq \sqrt{n}$. The cell $g_{i,j}$ is in the i th row and j th column. The expected number of points in each cell is 1. For any input point p , define its k -neighborhood to be a circle of radius k units centered at p . For example, for every input point p , we can find all the input points that are within a C -neighborhood of p in $O(n)$ time (C being a constant). The idea is to: 1) For each point p find the cell coordinates (i, j) of p (i.e., p is in $g_{i,j}$); 2) Sort the points according to their cell coordinates (using radix sort); For each cell $g_{i,j}$ let $L_{i,j}$ be the list of points in $g_{i,j}$. 3) To identify the points in a C -neighborhood of p , just examine the appropriate ($O(1)$) cells surrounding $g_{i,j}$.

Algorithm EMST

Let V be the set of given input points. $E := \emptyset$.

for every point $p \in V$ **do**

Identify the set Q of points that are within a distance of

$\frac{C_1}{\sqrt{n}}$ from p (for some constant $C_1 > 1$).

Q is identified using the grid that has been created.

for every $q \in Q$ **do**

Add (p, q) to E .

Employ any algorithm to identify the spanning forest F of $G(V, E)$. Let $T(V', E')$

be the largest tree (i.e., the tree with the largest number of nodes) in F .

Also, let $V'' = V - V'$ and $r = \frac{C_2 \log n}{\sqrt{n}}$, C_2 being a constant > 1 .

repeat

Let $E_1 = E'$.

for every point $p \in V''$ **do**

Identify the set R of points that are within a distance of r from p .

for every $q \in R$ **do**

Add (p, q) to E_1 .

Find the minimum spanning forest F of $G_1(V, E_1)$. If F has only one tree T_1 , output T_1 and quit; else let $r := 2r$.

forever

The correctness of the above algorithm is quite clear. We now show that the above algorithm runs in an expected $O(n)$ time. The following two Lemmas enable us to prove this.

Lemma 2.1 *The size of V'' is $O(n^\beta)$ for any constant $\beta > 0$.*

Proof. Suppose we use Prim's algorithm to find a minimum spanning forest of $G(V, E)$. Corresponding to $T(V', E')$, the algorithm will proceed as follows. It grows a tree T' starting with the minimum weight edge e of T . It examines all the outgoing edges from e and includes the minimum weight outgoing edge, and so on. Let q be the number of nodes in the tree T' at some point in time. The number of cells occupied by these q points is at most q and at least $q - \sqrt{b\alpha q \log n}$ with probability $\geq (1 - n^{-\alpha})$ for some constant b . Consider the C -neighborhoods of points in T' . The number of free cells in this neighborhood is at least $b'\sqrt{q}$ w.h.p. for some constant b' . (This happens when the points in T' occupy all the cells in a square of size $\sqrt{q} \times \sqrt{q}$). Probability P that none of these cells has an input point is $\left(1 - \frac{1}{n - q + O(\sqrt{q \log n})}\right)^{(n-q)b'\sqrt{q}}$. (This can be seen as follows. There are q points in T' and these points cover at least $q - o(q)$ cells. Thus the remaining $n - q$ points are found

(randomly) in the remaining (at most) $n - q + o(q)$ cells.) Probability P will be $\leq n^{-\alpha}$ as long as $q = \omega(\log^2 n)$ and $n - q = \omega(\log n)$. Under these conditions on q , at least one node of T' will have a (non-tree) neighbor within its C -neighborhood with high probability. Thus Prim's algorithm will be able to add another edge to T' . In other words, the edges in G are enough to grow a tree of size q where $n - q = \omega(\log n)$. \square

Lemma 2.2 *If $V'' = O(n^\beta)$ for some constant $\beta < 1$, then the algorithm EMST runs in an expected $O(n)$ time.*

Proof. Let p be any point in V'' . The expected number of input points in a $C_2 \log n$ -neighborhood (for some constant C_2) of p is at least $C_3 \alpha \log^2 n$ with probability $\geq (1 - n^{-\alpha})$ for some constant C_3 . Of these many points the probability that at least one point belongs to the tree $T(V', E')$ is very high (since V' has $n - O(n^\beta)$ points). Thus every point p in V'' has at least one node of T in its $C_2 \log n$ -neighborhood with high probability. As a result, the **repeat** loop of EMST runs only once with high probability. Also note that the additional number of edges added to E_1 in the **repeat** loop is $O(n^\beta \log^2 n) = o(n)$. \square

Lemmas 2.1 and 2.2 yield the following Theorem.

Theorem 2.1 *Algorithm EMST runs in an expected $O(n)$ time.*

3 Extensions

The algorithm of the previous section can be extended to higher dimensions as well. Consider a space of dimension k , where k is a constant. Assume without loss of generality that we are given n input points in a unit cube. We can partition the cube into subcubes of dimension $\frac{1}{n^{1/k}} \times \frac{1}{n^{1/k}} \times \cdots \times \frac{1}{n^{1/k}}$ each. The expected number of points in any subcube is only $O(1)$.

We can use the same idea and construct a graph $G(V, E)$ with $O(n)$ edges. If we examine the C_1 -neighborhoods of the input points to construct E , then the number of subcubes (or cells) examined is $(2C_1 + 1)^k$. Thus it takes $O(n(2C_1 + 1)^k)$ time to construct G . The same

amount of time is taken to solve the minimum spanning forest problem on G . In the **repeat** loop, the additional number of edges added to E_1 is $O(n^\beta(2C_2 \log n + 1)^k)$. Thus the run time of the entire algorithm will be $O(n(2C_1 + 1)^k + n^\beta(2C_2 \log n + 1)^k)$. As a result we get the following Theorem.

Theorem 3.1 *Algorithm EMST can be employed for higher dimensions as well. The algorithm runs in an expected $O(n(2C_1 + 1)^k + n^\beta(2C_2 \log n + 1)^k)$ time when the input points come from a Euclidean space of dimension k . Thus when k is a constant, the expected run time is $O(n)$.*

4 Conclusions

In this note we have pointed out that the k -dimensional EMST problem can be solved in an expected $O(n)$ time where n is the number of input points, as long as $k = O(1)$.

References

- [1] B. Chazelle, A minimum spanning tree algorithm with inverse-Ackerman type complexity, *Journal of the ACM* 47(6), 2000, pp. 1028-1047.
- [2] E. Horowitz, S. Sahni, and S. Rajasekaran, *Computer Algorithms*, W. H. Freeman Press, 1998.
- [3] D. R. Karger, Philip N. Klein, and R. E. Tarjan, A randomized linear-time algorithm to find minimum spanning trees, *Journal of the ACM* 42(2), 1995, pp. 321-328.
- [4] R. C. Prim, Shortest connection networks and some generalizations, *Bell System Technical Journal* 36, 1957, pp. 1389-1401.
- [5] M. I. Shamos and D. J. Hoey, Closest-point problems, *Proc. 16th Annual IEEE Symposium on Foundations of Computer Science*, 1975, pp. 151-162.

- [6] A. Yao, On constructing minimum spanning trees in k -dimensional spaces and related problems, *SIAM Journal on Computing*, 11(4), 1982, pp. 721-736.