SLAC PUB-1665
STAN-CS-75-529
Revised March 1977

FAST ALGORITHMS FOR CONSTRUCTING MINIMAL SPANNING TREES IN COORDINATE SPACES

Jon Louis Bentley

Department of Computer Science
University of North Carolina at Chapel Hill
Chapel Hill, N. C. 27514

and

Jerome H. Friedman *

Stanford Linear Accelerator Center
Stanford University, Stanford, Ca. 94305

ABSTRACT

Algorithms are presented that construct the shortest connecting network, or minimal spanning tree, of N points embedded in k-dimensional coordinate space. These algorithms take advantage of the geometry of such spaces to substantially reduce the computation from that required to construct minimal spanning trees of more general graphs. An algorithm is also presented that constructs a spanning tree that is very nearly minimal with computation proportional to N log N for all k.

(Submitted to IEEE Transactions on Computers)

*Work supported in part by Energy Research and Development Administration under contract EY-76-C-03-0515

INDEX TERMS

minimal spanning trees

graphs

clustering

shortest connection networks

trees

communication networks

I. Introduction

A transportation or communication network is composed of a set of nodes (or terminals) and a set of distances between pairs of nodes (the edges or branches of the network). The minimal spanning tree (MST) of such a network is a subset of the branches that have minimum total distance while providing a route between every pair of nodes.

The MST problem can be formally stated in graph theoretical terms. Consider a connected, undirected graph, G, with vertex set, V, and edge set, E (E is a subset of V & V); a spanning tree is a subset of E, such that there is a unique path between any two vertices in V. Suppose there is a cost associated with every edge in E; a minimal spanning tree of G is a spanning tree of G that minimizes the sum of the costs of the edges.

We present algorithms for finding minimal spanning trees for collections of points in multidimensional coordinate spaces. Here, the vertices of the graph are the points, and the cost of an edge between two nodes is the distance in the space between them. The graph representing such a collection of N points is a complete linear graph and has N(N-1)/2 edges.

This problem arises in many applications. Building a telephone network for many cities might call for finding the MST of a set of points in a two-dimensional Euclidean coordinate space. (See, for example, Prim[1]). Loberman and Weinberger[2] show that MSTs can be used to make optimal electric connections between a number of terminals on a breadboard. Zahn,[3,4] and Clark and Miller[5] apply MSTs to a wide variety of problems in pattern recognition. Arkadev and Braverman,[6] Johnson,[7] Gower and Ross,[8] and Zahn[3] apply MSTs to hierarchical clustering of points in multidimensional data spaces. Lee, Slagle and Blum[9] use the MST of points in multidimensional spaces for finding good nonlinear mappings of those points to two-dimensional spaces.

The MST problem has been extensively studied in the graph theoretic formulation. The classical algorithms for use with dense graphs have been given by Kruskal[10] and Prim.[1] Their efficient implementation on a computer was described by Dijkstra.[11,12] For sparse graphs (in which only few pairs of nodes have edges between them, or equivalently there are many edges of infinite cost), Yao[13] has given a fast MST algorithm. All of these algorithms can be applied to arbitrary graphs. Since the graphs defined by N points in a coordinate space are complete linear graphs with N(N-1)/2 edges, they are as dense as possible (for graphs of multiplicity - one), and Yao's algorithm does not result in better running times for these cases.

Because of their generality, these algorithms, when applied to points in coordinate spaces, do not take advantage of the geometry of the spaces to reduce computation. For example, if a human (rather than a computer) is constructing an MST on a distance true map of the 1000 most populous cities in the United States, he never considers an edge from New York to Los Angeles; their distance is clearly too great. Prim's method was capable of building a planar MST "based on visual judgments of relative distances, perhaps augmented by a pair of dividers in a few close instances".[1] Since a human is able to use such geometric considerations to advantage when constructing MSTs (at least in two and three-dimensional spaces), one might hope for computer algorithms that do so as well. Shamos[14] describes the application of the Voronoi diagram (a general structure for searching the plane) to the MST problem. He gives a worst case O(Nlog N) algorithm for finding the MST of a collection of N points in the two-dimensional plane with a Euclidean distance measure. Unfortunatly, this method has not yet been generalized to higher dimensionalties or more general distance measures.

We present algorithms that find MSTs in k-dimensional spaces using arbitrary distance or dissimilarity measures.[15] The algorithm is a variant of Prim's[1] that employs the fast nearest neighbor algorithm of Friedman, Bentley, and Finkel[16] to exploit the geometry of coordinate spaces. The performance of the algorithm is difficult to describe analytically, and depends somewhat upon the configuration of the points in the coordinate space. Monte Carlo simulations indicate that for many point configurations the computation is very nearly proportional to Nlog N for all dimensionalities and distance measures. A worst case exception occurs when the points are equally distributed among a few very widely separated clusters. For these cases, the rate of growth of the computation with number of nodes is slightly faster. In all cases, however, this algorithm is much faster than the general MST algorithms, except for very small data sets. A variant of the algorithm is presented that constructs spanning trees that are very nearly minimal with computation proportional to Nlog N for all point configurations.

2. Single Fragment Algorithm

Prim[1] suggested several definitions and proposed two principles for constructing minimal spanning trees. An isolated node is a node to which, at a given stage of construction, no connections have yet been made. A fragment is a node subset connected by edges between members of the subset. (A fragment is called a subtree in graph theory.) The distance of a node from a fragment, of which it is not an element, is the minimum of its distances from the individual nodes comprising the fragment. A nearest neighbor of a node is a node whose distance from the specified node is at least as small as that of any other. A nearest neighbor of a fragment is a node whose distance from the specified fragment is at least as small as that of any other.

With these definitions, Prim[1] enunciates two construction principles for minimal spanning trees:

Principle 1 - Any isolated node can be connected to a nearest neighbor.

Principle 2 - Any fragment can be connected to a nearest neighbor by a shortest available link.

Prim shows that if N-1 links are added in accordance with these principles, those links will form an MST for the nodes. A great many different algorithms are possible by combining these two principles in different ways. A human constructing an MST on a distance true map will generally choose the particular combination that tends to be relatively efficient for the particular configuration of points.

Both Prim[1] and Dijkstra[11,12] suggest that a single fragment algorithm is most efficiently implemented on a digital computer. This algorithm uses Principle 1 only once to produce a single fragment, which is then extended by N-2 applications of Principle 2.

The computation involved in selecting the nearest neighbor to the fragment can be reduced in two alternate ways. The first is to store for each node in the fragment, at a particular stage of construction, the identity of and distance to its closest isolated node. The second is to store for each isolated node the identity of and distance to its nearest neighbor within the fragment. These potential MST edges are called links. When a new node is added to the fragment, these links must be updated.

For the second alternative, this updating consists of calculating the distance from each isolated node to the new fragment node and seeing if it is smaller than the previous distance of the point to the fragment. If there are M nodes in the fragment, this updating requires computation proportional to N-M, for M going from 1 to N-1. Thus, the total computation is proportional to $N(N-1)/2$. With the first alternative, the up-

dating involves finding the nearest neighbor to the new fragment node among the isolated nodes. It also involves finding the nearest neighbors among the isolated nodes to those fragment nodes that previously had the new fragment node as their nearest isolated neighbor. If the nearest neighbors are found by calculating the distances to each isolated node and taking the smallest, the computation for each updating is proportional to $m(N-M)$, for $1 \leq m \leq M$, and M going from 1 to N-1. Here m is the number of nearest neighbor calculations required. Its average value depends upon the configuration of the points in the coordinate space. The total computation required for the first alternative ranges from a best case of $N(N-1)/2$ (m=1 for all cases) to a worst case of $N^2(N-1)/6$ (m=M for all cases). Under these premises, then, the second alternative (as suggested by Prim and Dijkstra) is clearly preferred.

There can be situations, however, where the first strategy is best. This would be the case if the nearest neighbor calculations did not require the examination of all isolated nodes, and if all m nearest neighbor calculations were not required when a new node is added to the fragment. Friedman, Bentley and Finkel[16] present an algorithm and data structure, called the k-d tree (see Appendix), for finding nearest neighbors to points in coordinate spaces, with computation proportional to log N after a pre-process (building the k-d tree) requiring NlogN. In most cases, this algorithm examines only a small fraction of the point set when finding nearest neighbors.

Nearest neighbor searches, themselves, can be avoided by arranging the nodes of the fragment in a priority queue. A priority queue is defined[18] to be a largest in, first out list. The priority of a fragment node is inversely related to the distance of its nearest isolated neigh-

bor. Therefore, the fragment node with the closest nearest isolated neighbor has highest priority. According to Prim's second MST construction principle, the link corresponding to the front of this queue is the one to be added to the fragment, as well as the node that is connected to the fragment by this link. The nearest isolated neighbor to this newly added fragment node must be found and entered into the priority queue. However, it may not be necessary to find new nearest neighbors for all those fragment nodes that previously had this new one as their nearest isolated neighbor. This is because the current priority of each of these nodes (the priority before the new one was added) serves as an upper bound on its real priority (the priority after the new one was added). The real priority of a fragment node need only be found when its upper bound (current priority) is highest in the queue. In this manner, a great many nearest neighbor searches can be avoided. Priority queues implemented either as heaps[17] or leftist trees[18] can be manipulated with computation proportional to the logarithm of their size.

A single fragment algorithm for constructing an MST for a collection of points in a coordinate space would proceed as follows:

Single Fragment Fast MST Algorithm

```
Build k-d tree for the point set;
Pick an arbitrary point as first node;
Find its nearest neighbor;
Establish link to nearest neighbor;
Enter node and its (real) priority into priority queue;
Loop until MST completed (N-1 nodes added to fragment);
    Loop until highest priority in queue is real;
        X ← top node in queue;
        Y ← nearest isolated neighbor to X;
        Establish link from X to Y;
        Replace X's upper bound with real priority;
        Re-insert X into priority queue with real priority;
    Repeat;
    X ← top node of queue;
    Y ← isolated node linked to X;
    Insert (X,Y) link as edge in MST;
    Find nearest isolated neighbor to Y;
    Establish link to Y's nearest isolated neighbor;
    Enter Y and its (real) priority into queue;
Repeat;
```

Since the nearest neighbor searches and priority queue insertions can be performed in O(log N)time, this algorithm requires computation proportional to m log N for each link insertion. Here m is the number of nearest neighbor searches required to add the link (that is, the number of executions of the inner loop plus one). The total computation is proportional to $(N-1)\bar{m}$ log N, where $\bar{m}$ is the average of m over all the links added to the MST. Thus, if $\bar{m}$ is small compared to N/(2log N), one might expect this algorithm to have computational advantage over the more general MST algorithm of Prim and Dijkstra.

3. Multifragment Algorithm

The speed of the single fragment algorithm depends directly on the value of $\bar{m}$, which depends upon the configuration of the points in the coordinate space. It also depends upon the way the fragment grows; that is, the order in which the links are added to form the complete MST. Zahn[3] has pointed out that the MST tends to follow density gradients of the point set. Prim's construction principles ensure that once a fragment is started, its growth will follow the local density gradients "uphill" until it reaches a local density maximum. Once it reaches that maximum, it will tend to add links in directions of minimum decreasing density, slowly growing downhill.

The number of nearest neighbor searches for each link added to the fragment depends strongly on whether the fragment is growing toward increasing or decreasing density. If the fragment is growing toward increasing density, then each new node added to the fragment tends to have a shorter nearest neighbor link than those already in the fragment, and its priority will be at or near the top of the fragment priority queue. Since the priority of a newly added node is always real (rather than an upper bound), it will quickly

become a candidate for a new fragment edge without the necessity of updating the upper bounds for those nodes lower in the queue. It is this updating that is performed by the inner loop of the algorithm and contributes all but one of the m nearest neighbor searches required to add the link.

The situation is completely reversed when the fragment is growing toward decreasing density. Then each new link added to the fragment tends to be longer than those previously added, and its corresponding low priority sinks it to a low position in the priority queue. As a result, it is more likely that a node with only an upper bound for its priority is at the top of the queue. Thus, several near neighbor searches could be required before a node with a real priority reaches the top of the queue.

Many point distributions encountered in practice tend to have small, localized regions of high density and broad, diverse regions of low density. Even if a fragment happens to start in a low density region (which is unlikely with a random choice since most of the points are in regions of high density), it will quickly reach the maximum in a few steps. It will then add most of its links while going downhill, thereby incurring large values of m with each added link.

It is possible to greatly increase the speed of the algorithm by employing Prim's construction principles in a multiple fragment strategy. The first fragment is started with the lowest density point (the point for which the local density is lowest). It then grows uphill following density gradients until it reaches a local density maximum. At that time, its growth is terminated. A new fragment is then started at the isolated node with the lowest local density. This fragment grows uphill until it either

joins with an already existing fragment (at which time it is merged with that fragment) or until it reaches a density maximum. This procedure continues until there are no isolated nodes remaining. At that point, there exist one fragment or several unconnected fragments. Since each of these fragments were constructed using Prim's construction principles (single fragment fast MST algorithm), they are all subtrees of the complete MST.

The fragments must be connected to form the complete MST. If there are L unconnected fragments, then there are L-1 links remaining to complete the MST. These links can be found by resuming the growth of the smallest fragment at that point where it was previously terminated. The first link that it adds will be the MST edge connecting it to another fragment. These two fragments are merged and the process is repeated with the smallest fragment[19] that now remains (after the merge). This merging procedure is repeated until all fragments are connected and the MST is complete.

Implementation of this multifragment strategy requires a density estimation for each point in the coordinate space and a criterion for stopping fragment growth in dense areas. The purpose of the density estimation is simply to start fragments in regions of relatively low density and need not be very accurate. This density estimation is provided by the k-d tree data structure (see Appendix) used for the fast nearest neighbor searches. This data structure partitions the coordinate space into cells that each contain very nearly the same number of points. The number of points in a cell, divided by its volume, can serve as a local density estimate for the points in the cell. Each point will then have an associated density estimate.[20]

The purpose of stopping fragment growth is to prevent the number of

nearest neighbor calculations, m, per added link, from becoming large. The most straightforward way to accomplish this is to terminate when the value of m for a given link exceeds some pre-specified maximum, m_o. For the reasons described above, this will tend to happen when the fragment has reached a density maximum and starts to grow toward decreasing density.

The full multifragment algorithm for constructing an MST for a collection of points in a coordinate space is as follows:

Multiple Fragment Fast MST Algorithm

```
Set value for parameter m_o;
Build k-d tree for point set, and calculate density estimates;
Sort points in ascending order of density estimates;
Loop until no isolated points are left;
    Pick isolated point of lowest density to start a new fragment;
    Find its nearest neighbor;
    Establish link to nearest neighbor;
    Enter node and its (real) priority into queue for this fragment;
    Loop until m > m_o for adding a link or MST finished
                                        (N-1 links added);
        m ← 0
        Loop until highest priority in queue is real or m > m_o;
            X ← top node in queue;
            Y ← nearest non-fragment neighbor to X;
            Establish link from X to Y;
            Repace X's upper bound with real priority;
            Re-insert X into priority queue with real priority;
            m ← m + 1;
        Repeat;
```

```
            If (m > m_o) then EXIT LOOP;
            X ← top node of queue;
            Y ← node linked to X;
            Insert (X,Y) link as edge in fragment;
            If (Y belongs to another fragment) then
                 Merge the two fragments;
                 EXIT LOOP;
            End if;
            Insert Y into fragment;
            Find nearest non-fragment neighbor to Y;
            Establish link to Y's nearest non-fragment neighbor;
            Enter Y and its (real) priority into fragment queue;
       Repeat;
  Repeat;
  Loop until one fragment remains;
       Find fragment of lowest cardinality;
       Loop until highest priority in fragment queue is real;
            X ← top node in queue;
            Y ← nearest non-fragment neighbor to X;
            Establish link from X to Y;
            Replace X's upper bound with real priority;
            Re-insert X into queue with real priority;
       Repeat;
       X ← top node of queue;
       Y ← node linked to X;
       Insert (X,Y) link as edge in fragment;
       Merge the two fragments connected by the link;
  Repeat;
```

The sort of the N points (on the basis of their density estimates) can be accomplished in O(N log N) time.[18] As with the single fragment algorithm, the nearest neighbor searches and priority queue insertions can be performed in O(log N) time. Implementing the priority queues as leftist trees allows them to be merged in O(log N) time.[18] Therefore, the total computation with this multifragment algorithm is proportional to $(N-1)\bar{m}$ log N. However, because most of the links have been added while fragments were growing toward increasing density, the value of $\bar{m}$ will be considerably smaller than that for the single fragment algorithm.

The rate of increase of $\bar{m}$ with increasing number of nodes N determines the rate of growth of computation time for the algorithm. During the fragment growing stage of the algorithm, $\bar{m}$ is independent of N since it is bounded by m_o, so the time required to build the fragments is proportional to Nlog N.

Since fragment growth is terminated in high density regions and these regions tend to be highly localized in the coordinate space, the links that connect fragments are usually small compared to most links within the fragments. The real priorities corresponding to these links will be high in the fragment queue so that generally few nearest neighbor searches are required to link the fragments. Thus, the computation required to construct the MST for this multifragment algorithm is usually dominated by the Nlog N time required to build the fragments.

There is a special worst case situation where connecting the fragments can dominate the computation for the multifragment algorithm. This occurs when the distribution of points in the coordinate space is comprised of a very few compact clusters separated by large distances, with each cluster containing a substantial fraction of the total number of points.

A basic assumption of the logarithmic time nearest neighbor search algorithm is proximity; that is, that the distance to the resulting nearest neighbor is relatively small. Specifically, the volume of a ball with that distance as its radius must be small compared to the total volume occupied by all the points in the coordinate space. For an unrestricted nearest neighbor search, this is always true. However, the nearest neighbor searches performed here are restricted to those points outside the particular fragment under consideration. If the fragment to be linked is a large distance from all of the other points, then the distance to the closest point outside the fragment will not be small, and the time required to find it will grow more rapidly with N than log N. Therefore, the time required to link the final unconnected fragments for this case can grow faster than N log N.

4. Simulation Studies

In order to help quantify the above discussions and obtain a feeling for the performance of the multifragment algorithm, the results of several simulation experiments are presented. Figures 1 and 2 compare the running times, under identical conditions, of this special algorithm (multifragment) to the more general algorithm of Dijkstra (as published in FORTRAN by Whitney)[21] for several situations.[22] Although these relative running times may be somewhat installation dependent, they can give at least a general idea of the comparative properties. These figures show the running time per node as a function of total number of nodes on a logarithmic scale. The solid squares in Figure 1 show results for spherically symmetric, normally distributed points in a two-dimensional coordinate space with a Euclidean distance measure. The running time per node is clearly logarithmic for this case; thus, the total running time is proportional to N log N. The open circles are the results for a point distribution

comprised of two spherical normals with unit covariance matrix where the centers are separated by a distance of ten. Here, the running time per node is seen to rise somewhat faster than logarithmic for large numbers of nodes. The open squares represent the corresponding running times under the same conditions for the general MST algorithm of Dijkstra.[23] This general MST algorithm is seen to be considerably slower than the one presented here (multi-fragment), except for very small node sets. Figure 2 presents results for a spherical normal distribution in five dimensions, with Euclidean distance measure. The running time per node is seen to be clearly logarithmic for this case as well.

As discussed above, the value of $\bar{m}$ for a particular situation depends upon the distribution of the points in the coordinate space. In the simulations presented here, the value of $\bar{m}$ ranged from 1.3 to 3.

Comparison of Figures 1 and 2 indicates that the point at which the special MST algorithm obtains computational advantage over the general algorithm depends upon dimensionality of the coordinate space. This is a consequence of the fact that the relative time required to find nearest neighbors increases with dimensionality of the coordinate space. Table 1 lists the total number of nodes at which the special MST algorithm becomes faster than the general algorithm.[24]

5. A Fast Almost Minimal Spanning Tree (AMST) Algorithm

For the worst case situation, where the nodes are concentrated in a few well separated clusters, the multifragment algorithm spends most of its computation finding the few MST links that connect these clusters. In many applications, these precise links are not required. In clustering applications, for example, the lengths of these links are used to establish the existence of the clusters, and then the links are deleted.

Since the multifragment algorithm has established the existence of these clusters before it adds these final connecting links, it is unnecessary to add them. In all statistical applications, and in most geometrical applications, the precise minimal spanning tree is not required. A spanning tree that is very nearly minimal will serve just as well.

It is possible to avoid the increased computation for the worst case situation by detecting when it is occurring and then shifting to a faster approximate strategy.

Prim's second construction principle, which is used to connect the fragments, insures that they will be connected by a link that represents the smallest interpoint distance between the fragments. This insurance can be quite costly in worst case situations. For these situations, the intercluster links tend to be longer than most (if not all) intracluster links. The priorities in the fragment queue will be mostly upper bounds, and these upper bounds will be higher in the queue than the interfragment links representing real priorities. Thus, a great many nearest neighbor searches are required before a real priority reaches the top of the queue. In the very worst case, near neighbor searches are required for all points in the fragment. As described above, these searches do not obey the proximity principle required by the fast near neighbor algorithm so that the time required for them is slower than logarithmic.

This problem can be sidestepped by pursuing a different strategy for finding smallest interpoint distances between fragments. Since this strategy does not guarantee that the link it finds will be absolutely the smallest, Prim's construction principles may be violated. It can guarantee, however, that if the link it finds is not the smallest, its length is nearly the same as the smallest one. Thus, the resulting spanning tree is very nearly minimal.

This strategy proceeds as follows: the fragment priority queue is scanned for its smallest interfragment link. The endpoint of this link in the other fragment becomes a focal point and the endpoint in the current fragment becomes its corresponding point. The nearest non-fragment neighbor to the focal point is found.

If this neighbor is not its corresponding point, the neighbor becomes the focal point and the former focal point becomes its corresponding point. This procedure is iterated until the nearest neighbor to the focal point becomes its corresponding point. The link between these two points is then taken as the edge connecting the two fragments in the spanning tree.

The decision to invoke this approximate strategy should be taken only when a worst case situation (clustering) is detected. A signature for clustering is that the MST edge from one fragment to another has not been found after more than a few nearest neighbor searches. The approximate strategy is thus invoked when the number of nearest neighbor searches, n, required to connect the fragment to another fragment, exceeds some prespecified cutoff, n_o.

An almost minimal spanning tree algorithm proceeds as follows:

Multifragment Almost Minimal Spanning Tree Algorithm

```
Set value for parameter n_o;
Construct fragments as with multifragment MST algorithm;
Loop until one fragment left;
    Find smallest fragment;
    n ← 0;
    Loop until highest priority in queue is real or n > n_o;
        X ← top node in queue;
        Y ← nearest non-fragment neighbor to X;
        Establish link from X to Y;
        Replace X's upper bound with real priority;
        Re-insert X into queue with real priority;
        n ← n + 1;
    Repeat;
    If (highest priority in queue is real) then
        X ← top node of queue;
        Y ← node linked to X;
        Insert (X,Y) link as edge in fragment;
        Merge two fragments connected by the link;
    Else
        Find closest existing link in queue to another fragment;
        X ← associated node in fragment;
        Y ← associated node in other fragment;
        Loop;
            Z ← nearest non-fragment neighbor to Y;
            If (Z = X) then EXIT;
            X ← Y;
            Y ← Z;
        Repeat;
        Establish link from X to Y;
        Insert (X,Y) link as edge between fragments;
        Merge two fragments connected by the link;
    End if
Repeat;
```

The AMST algorithm constructs the true minimal spanning tree in all but the worst case situations. In those situations, it tries to find the shortest connecting link by means of the heuristic strategy described above.

Since many points in one cluster have in common the same closest point in the other, the rate of convergence of the iterative search in the heuristic strategy is quite fast and is independent of the total number of nodes, N.[25] Each iteration requires a nearest neighbor calculation which, at worst, requires time proportional to N. Therefore, the time required to link fragments with the approximate strategy is proportional to N. The total time to construct the AMST is then dominated by the fragment building which requires computation proportional to Nlog N.

In order to obtain a feeling for the performance of the AMST algorithm in worst case situations, Table 2 gives the results of several Monte Carlo simulation experiments. Here 1000 points were sampled from five four-dimensional spherical normal distributions with unit covariance matrix, each centered at a different vertex of a four-dimensional simplex with edge length ten. Table 2 compares the total length of the AMST to the length of the true MST, as well as the running time for each. Five independent samplings are presented. It is seen that for these experiments, the length of the AMST was, at most, 0.16% longer than the true MST, but took less than half as much running time to construct.

6. Storage Requirements

This section discusses the memory requirements for the MST algorithms presented here and compares them to the requirements of the general algorithm of Prim and Dijkstra. For all MST algorithms, the coordinates of each point must be stored for input, and the resulting MST must be stored

for output. The coordinates of the points require k·N real numbers where k is the dimensionality of the coordinate space. The MST requires N-1 real numbers to store the edge lengths and 2·(N-1) integers to store the node identities defining the edges. The Prim-Dijkstra algorithm requires, in addition, N real numbers and 2·N integers. Thus, the Prim-Dijkstra algorithm requires storage for (k+2)·N real numbers and 4·N integers.

The additional storage required by the single fragment fast MST algorithm includes N/16 real numbers and (1+1/16)·N integers to store the k-d tree, N real numbers and N integers to store and manage the priority queue, and N integers to store temporary pointers to isolated nodes. Thus, the single fragment algorithm requires a total storage of (k+2+1/16)·N real numbers and (5+1/16)·N integers.

The multiple fragment fast MST algorithms require additional storage beyond that for the single fragment algorithm. This includes 4·N integers to implement the priority queues as leftist trees and 2·N integers to handle the fragment bookkeeping. Thus, the multiple fragment algorithms require a total storage of (k+2+1/16)·N real numbers and (11+1/16)·N integers.

In all cases, the storage requirement is linear in the number of nodes. The increase in storage for the fast MST algorithms over that of the Prim-Dijkstra algorithm is not severe. If an integer requires half as much storage as a real number, then in a two-dimensional space the increase is ten percent for the single fragment algorithm and sixty percent for the multiple fragment algorithms. In eight dimensions, the corresponding numbers are five percent and thirty percent.

7. Discussion

Three algorithms have been presented for constructing MSTs in coordinate spaces. The single fragment algorithm has the advantage of simplicity, smaller memory requirement, and it requires no external parameters to be specified. It is also slightly faster than the other algorithms for point distributions that are nearly uniform.[26] It has the disadvantage that the number of nearest neighbor searches required to add each MST edge can be quite large. In fact, a worst case situation could arise where a nearest neighbor search is required for every node currently in the fragment. For this worst case, the time to build the MST with this algorithm is $O(N^2 \log N)$, which is worse than $O(N^2)$ required by the general MST algorithm of Prim and Dijkstra.

The multifragment algorithm is considerably faster than the single fragment algorithm for non-uniform distributions. It requires the specification of a single external parameter, m_o; this is the maximum number of nearest neighbor searches permitted for adding a link to a fragment. If this number is exceeded, fragment growth is terminated and a new fragment started. Experience has shown that the running time of the algorithm is insensitive to the value chosen for m_o. Values of m_o from 5 to 15 work well.

The AMST algorithm is identical to the multifragment MST algorithm except for the special worst case clustering situations. For these situations, the AMST algorithm is considerably faster than the true MST algorithms, but it may not yield the exact MST. The spanning tree that it does yield, however, will be almost minimal. The AMST algorithm requires the specification of an additional parameter. This is the maximum number of nearest neighbor searches, n_o, allowed while trying to link a fragment

to another one, before the fast heuristic strategy is invoked. A choice for its value involves a trade-off between running time and the degree to which the AMST is required to match an exact MST. Values of n_o from $5\ m_o$ to $10\ m_o$ are reasonable.

The worst case behavior of the multifragment MST and AMST algorithms is not clear. It is surely much better than the $O(N^2 \log N)$ of the single fragment algorithm. A necessary (but not sufficient) condition for worst case behavior with the single fragment algorithm is that every edge added to the fragment must be longer than all those currently in the fragment. This means that the fragment is always growing towards decreasing density of points. This situation cannot occur with the multifragment algorithms since they start fragments in regions of low density and add links mainly in directions of increasing density.

8. Conclusion

Algorithms have been presented for constructing minimal spanning trees for complete graphs, where the nodes are points in k-dimensional coordinate spaces and the edge weights are general distances between them. The algorithms employ Prim's[1] construction principles in a way that allows the use of the fast (logarithmic) near neighbor algorithm of Friedman, Bentley, and Finkel[16] to reduce computation in finding nearest neighbors, and the use of logarithmic priority queues[17,18] to reduce the number of nearest neighbor searches. A multifragment strategy is presented that takes advantage of density gradients in the distribution of points to achieve a much greater reduction in nearest neighbor searches.

For N points embedded in a k-dimensional coordinate space, this algorithm will usually construct the MST in time proportional to Nlog N. Special worst case situations exist for which the time spent in connecting fragments dominates the computation, and although the algorithm is still much faster than earlier algorithms, it can require computation in excess of Nlog N. A method for detecting this situation and shifting to a fast heuristic strategy for connecting the fragments in this special case is presented. Although this heuristic strategy cannot guarantee that the links it finds are of minimum total length, they are always quite close. Thus, the resulting spanning tree that it constructs is almost minimal. This AMST algorithm constructs minimal or very nearly minimal spanning trees with average computation proportional to Nlog N. All of these algorithms are considerably faster than earlier algorithms that construct MSTs for general graphs, except when the graphs are small.

Acknowledgment

Helpful discussions with F. Baskett, C. T. Zahn and J. E. Zolnowsky are gratefully acknowledged.

APPENDIX

k-d Trees

For completeness, this appendix provides a brief description of the k-d tree data structure and the fast near neighbor search algorithm.[16] The general discussion presented here is quite limited and details can be found in Reference 16.

The k-d tree is a generalization of the simple binary tree for sorting and searching. The k-d tree is a binary tree in which each node represents a sub-collection of the points and a geometrical partitioning of that sub-collection. The root of the tree represents the entire collection. Each nonterminal node has two successors. These successor nodes represent the two sub-collections defined by the partitioning. The terminal nodes represent mutually exclusive small subsets of the points, which collectively form a geometric partition of the k-dimensional coordinate space.

At each nonterminal node, the geometrical partitioning is accomplished by dividing the sub-collection at the median value of one of the coordinates. The particular coordinate chosen to make the partition is the one which exhibits the greatest range or spread in values.

Associated with each node and the sub-collection which the node represents is a set of geometric bounds within which all points in that sub-collection must lie. These geometric bounds can be represented by two linear arrays LOWER and UPPER. For a given sub-collection, S, it must be true for every point $\vec{X} \in S$ and each coordinate i ($1 \leq i \leq k$) that $\text{LOWER}(i) \leq X(i) < \text{UPPER}(i)$. The values contained in these two arrays at any node in the k-d tree are determined by the partitions defined at its ancestors in the tree. Associated with each such ancestor nodes is a partitioning coordinate j and a partition value p. UPPER (j) is replaced by

p if the descent goes to the left son; otherwise LOWER (j) is replaced by p. (The bounds for the root node are initially set to plus and minus infinity, respectively.) These bounds jointly form a rectilinearly oriented hyper-rectangle in the k-dimensional coordinate space within which all of the points in any sub-collection must lie.

A nearest neighbor search employing the k-d tree can be easily described recursively. At each node visited in the search, the subtree representing the sub-collection of points on the same side of the partition as the test point is searched for its nearest neighbor in that sub-collection. When control returns, a test is made to see if the subtree representing the sub-collection on the opposite side of the partition must be searched. It must be searched only if the geometric bounds associated with that sub-collection overlap a ball centered at the test point, with radius equal to the current nearest neighbor distance. If the bounds do overlap the ball, then the opposite subtree is searched. In either case, a ball-within-bounds test is made before returning. This determines whether the nearest neighbor ball is completely within the geometric bounds of the node. If so, then the current nearest neighbor is correct for the entire collection of points and the search terminates.

It is shown in Reference 16 that the time required to build the k-d tree is O(Nlog N). The expected nearest neighbor search time is shown to be O(log N). These results are shown to be valid for arbitrary distributions of the points in the coordinate space and for a wide range of dissimilarity measures. In particular, it is not necessary that the dissimilarity measure be a metric distance and satisfy the triangle inequality.

Footnotes and References

1. R. C. Prim, "Shortest connection networks and some generalizations," Bell Syst. Tech. J. 36 (Nov. 1957), 1389-1401.

2. H. Loberman and A. Weinberger, "Formal procedures for connecting terminals with a minimum total wire length," Jour. ACM 4 (Oct. 1957), 428-437.

3. C. T. Zahn, "Graph-theoretical methods for detecting and describing gestalt clusters," IEEE Trans. Comput., C-20 (Jan. 1971), 68-86.

4. C.T. Zahn, "An algorithm for noisy template matching," IFIP Congress 1974, Vol. 4, American Elsevier Publ. Co., New York (1974), 698-701.

5. R. Clark and W.F. Miller, "Computer based data analysis systems at Argonne," Methods in Computational Physics, Vol. V.

6. A.G. Arkadev and E.M. Braverman, "Computers and Pattern Recognition," Washington, D.C., Thompson Book Co., 1967.

7. S. C. Johnson, "Hierarchical clustering schemes," Psychometrika, Vol. 32, (Sept. 1967), 241-254.

8. J. C. Gower and G. J. S. Ross, "Minimal spanning trees and single linkage cluster analysis," Appl. Statistics, Vol. 18, (1969) 54-64.

9. R. C. T. Lee, J. R. Slagle, and H. Blum, "A triangulation method for the sequential mapping of points from N-space to 2-space," IEEE Trans. Computers, C-26 (March 1977).

10. J. B. Kruskal, Jr., "On the shortest spanning subtree of a graph and the traveling salesman problem," Proc. Amer. Math. Soc., No. 7 (1956), 48-50.

11. E. W. Dijkstra, "A note on two problems in connexion with graphs," Numer. Math. 1, 5 (Oct. 1959), 269-271.

12. E. W. Dijkstra, "A short introduction to the art of programming," Technological University Eindhoven Report No. EWD 316 (Aug. 1971), 64-70.

13. A. Yao, "An O(|E| log log |V|) algorithm for finding minimal spanning trees," Info. Proc. Letters 4, 1 (1975), 21-23.

14. M. Shamos, "Computational Geometry," Ph.D Thesis, Yale University (1975).

15. These algorithms can be used with any generalized dissimilarity measure and, in particular, they do not require that it obey the triangle inequality.

16. J. H. Friedman, J. L. Bentley, and R. A. Finkel, "An algorithm for finding best matches in logarithmic expected time," Stanford Univ., Computer Science Report STAN-CS-75-482 (Feb. 1975).

17. J. W. J. Williams, ALGORITHM 232, HEAPSORT, Comm. ACM 7, 6 (June 1964), 347-348.

18. D. E. Knuth, The Art of Computer Programming, Vol. 3 Sorting and Searching, Addison-Wesley, Menlo Park (1973).

19. The smallest fragment is chosen in order to minimize fragment merging time.

20. The variance of these density estimates over the point set can be used as an indicator of whether there is sufficient density variation to justify the invocation of the multifragment algorithm.

21. V. K. M. Whitney, ALGORITHM 422, MINIMAL SPANNING TREE[H]. Comm. ACM 15, 4 (April 1972), 273-274.

22. All simulations for both algorithms were preformed on an IBM 370/168 computer. All programs were coded in FORTRAN IV and compiled with the IBM FORTRAN G compiler.

23. The running time for this algorithm is independent of the point distribution.

24. The results shown here are for spherical normal distributions and Euclidean distance measure. These values do not strongly depend upon the distribution of the points or on distance measure.

25. The actual rate of convergence depends on the geometrical shapes of the clustered points in the coordinate space. If the points in each cluster are divided into subsets, such that all points in the same subset have the same nearest neighbor outside the cluster, a sufficient condition for the independence of the convergence rate on N is that the number of such subsets be independent of N. For large enough N, the regions containing the subsets are defined by the probability density of the points and the geometry of the coordinate space, both of which are independent of N.

26. With the single fragment algorithm, once a node is entered into the MST it cannot later appear as a nearest non-fragment neighbor. Thus, it can be "poisoned" or removed from consideration in all subsequent nearest neighbor searches. When all points represented by a particular subtree of the k-d tree are poisoned, the root of that subtree can also be poisoned. This technique allows some reduction in the average time required to find nearest neighbors for the single fragment algorithm.

TABLE 1

Number of nodes, as a function of coordinate space dimensionality, beyond which the multifragment algorithm presented here is faster than Whitney's publication of Dijkstra's algorithm.[22]

Dimensionality	Crossover Point
2	250
3	260
4	340
5	445
6	645
7	920
8	1400

TABLE 2

Comparison of multifragment MST algorithm with AMST algorithm for a worst case situation.

Run Number	MST Length	AMST Length	AMST/MST-1 Length	AMST/MST Running Time
1	1022.1	1022.1	0	0.47
2	1045.0	1046.5	0.14%	0.44
3	1057.4	1059.1	0.16%	0.43
4	1022.9	1023.8	0.09%	0.48
5	1026.4	1027.5	0.11%	0.45

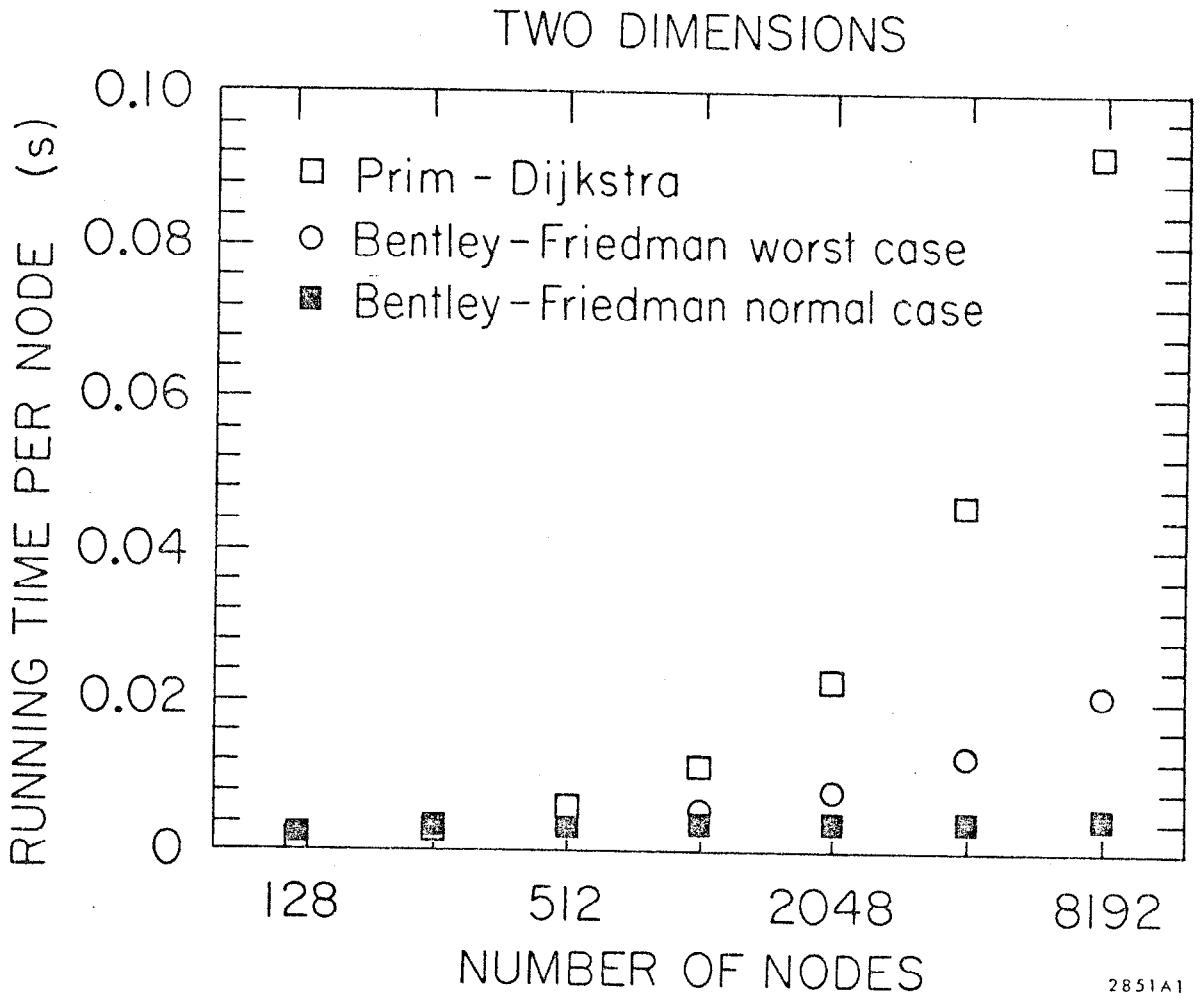
TWO DIMENSIONS
RUNNING TIME PER NODE (s)
0.10
0.08
0.06
0.04
0.02
0
Prim - Dijkstra
Bentley-Friedman worst case
Bentley-Friedman normal case
128
512
2048
8192
NUMBER OF NODES
2851A1

Fig. 1