Approximate single linkage cluster analysis of large data sets in high-dimensional spaces

William F. Eddy*, Audris Mockus, Shingo Oue

Department of Statistics, Carnegie Mellon University, Pittsburgh, PA 15213-3890, USA

Abstract

Motivated by a real-world data set of more than 40,000 observations, we consider single linkage clustering of large data sets in high-dimensional spaces. In this particular case, the data we consider are observations in the space of one-dimensional submanifolds of Euclidean space. This data set is so large that the individual observations cannot each be inspected. A goal of the analysis is to produce a partial ordering of the data so that a useful inspection of selected observations can take place. An essential step is construction of a metric on the objects in this high-dimensional space that can be computed fairly quickly. The size of the problem makes the running time of Prim's algorithm (the fastest available algorithm for finding a minimal spanning tree of a complete graph) prohibitive. We have developed several methods which approximate the single linkage tree. We report the result of applying our approximations to the data set.

1. Introduction and summary

The size and complexity of data sets we are presented for analysis is ever increasing. The recent increase in very large data sets (typically they are collected automatically) has stimulated new approaches to data analysis. The size and complexity of these data sets prevent the inspection of individual observations and forces us to rely more and more heavily on automatic procedures for the analysis of data.

Here we describe part of an example of exploration of a large high-dimensional data set. Other aspects of our exploration are described in Eddy and Oue (1994). We intend to use the ideas described here on other data sets.

* Corresponding author.

0167-9473/96/$15.00 © 1996 Elsevier Science B.V. All rights reserved
PII S0167-9473(96)00019-9
Our analysis is based on the single linkage clustering algorithm or the minimal spanning tree. We have chosen this method because

1. It is well understood;
2. For general input the results are invariant to the order of input;
3. The basic algorithm is nearly as fast as any algorithm for clustering.

An outline of the remainder of the manuscript is as follows. In the next section we describe the data set. Section 3 contains a description of the standard algorithms for finding a minimum spanning tree, Kruskal’s algorithm and Prim’s algorithm. We describe several methods which produce an approximate minimum spanning tree in Section 4. We indicate some of the results of applying our approximations to the data set in Section 5. We conclude with some discussion of this approach and how it might be used in other situations.

2. Description of the data set

The data we study here concern flights of aircraft controlled by the Federal Aviation Administration (FAA) over the United States during a single 24 h period on 25 October, 1991. Actually, the data cover the period from 3:00 Central Standard Time (CST) on 25 October, 1991 to 3:00 CST on 26 October, 1991 because nationally total air traffic reaches its daily minimum at approximately 3:00 CST. (All times in this paper are given on a 24 h clock set to CST.) The data set consists of 531,097 position reports on the controlled aircraft.

Each position report consists of the time of report, the position (latitude in hundredths of degrees, longitude in hundredths of degrees, and altitude), an encoded flight identifier, and nominal air speed. We have ignored air speed throughout this analysis. At approximately 5 min intervals, there is a report in our data set for each aircraft in the air. Longer intervals between position reports occurred when one or more of the regional air traffic control centers lost its communication link with the center where data consolidation took place.

If an aircraft flies from New York to San Francisco with a stop in Pittsburgh, it is assigned one identifier for the whole trip. However, for our purpose, this whole flight path is divided into two “legs”, i.e., New York to Pittsburgh and Pittsburgh to San Francisco. Each flight leg was assigned a unique identifier and sub-identifier pair. We identified 40,805 distinct flight legs in the data set. We treat each flight leg as a single observation in the space of one-dimensional manifolds in Euclidean three-space. Various preprocessing steps used to clean and organize the data set are described in Eddy and Oue (1994) and will not be repeated here. A natural way to think of these data is by thinking of the path of a single aircraft in space–time as a single observation. However, we do not know the whole flight path but rather several points on it.

We have learned, subsequently, that the data we obtained are known as “en route” reports, to be distinguished from reports obtained during take-offs and landings. This means that an aircraft does not appear in our data until several (up to
about ten) minutes after take-off and disappears from our data several minutes before landing.

3. Minimum spanning tree algorithms

We have chosen single linkage clustering as our approach to producing a partial order on the observations (an aid to selecting individual observations for inspection) simply because it is about the fastest available method for producing such an ordering with the property that its behavior for general inputs is invariant to the order of the input.

There are several algorithms for finding a minimum spanning tree given a graph with \( E \) edges on \( N \) vertices. The two best known and the ones we used are the algorithm of Prim (1957) and the algorithm of Kruskal (1956). A modern source for these and related algorithms is Sedgewick (1983). The worst-case running time for Prim's algorithm is \( O(N^2) \). The worst-case running time for Kruskal's algorithm is \( O(E \log E) \). The complete graph on \( N \) vertices has \( O(N^2) \) edges so Prim's algorithm is preferred for complete (and other dense) graphs.

Prim's algorithm is based on the fundamental property of minimum spanning trees: If the vertices of a graph are divided into two arbitrary subsets, the minimum spanning tree contains the shortest edge connecting a vertex in one set to a vertex in the other set. The algorithm begins by choosing one of the subsets to contain only one vertex (refer to this subset as the visited subset and the other as the unvisited subset). When the edge joining the two subsets is found the vertex on the end in the unvisited subset is transferred to the visited subset. This process is iterated until all vertices have been transferred to the visited subset.

Kruskal's algorithm simply sorts all the edges in the complete graph by their length. Then beginning with the shortest, the edges are examined one at a time and added to the minimum spanning tree whenever that addition does not create a cycle in the graph.

Unfortunately, for a complete graph on a large number of vertices (in our example, around 40,000) and a complicated distance function the computational time of even the quadratic algorithm is too long. For our data set, Prim's algorithm with a trivial (and useless) distance function took 30 min on a very fast workstation (Hewlett-Packard 9000/735). With a more realistic distance function it took almost eight days of computation. To speed up the calculations we chose to look for approximate solutions.

4. Approximation algorithms

To reduce computational costs we will produce several approximations to the minimum spanning tree and then will try to select one that balances speed and accuracy. The basic idea is to map the objects from the high-dimensional metric space into a two-dimensional space with the Euclidean metric. We will use the
Delaunay triangulation (the dual of the Voronoi tessellation) for this purpose. The Voronoi tessellation of a set of \( n \) sites is a partition of Euclidean space into \( n \) convex regions, the region corresponding to a particular site contains all points of the space which are closer to the site than to any other site. The Delaunay triangulation is the dual to the Voronoi tessellation in the sense that each face of the tessellation is replaced by a perpendicular to the face which joins the two sites separated by the face.

In the two-dimensional space, the Delaunay triangulation (see, e.g., Fortune (1987)) could be computed, reducing the number of edges (as compared to a complete graph) that could possibly be in the minimum spanning tree. The minimum spanning tree can be constructed in linear time from a triangulation (see Klein and Tarjan (1993)); unfortunately, we did not have a program implementing that algorithm readily available. The Delaunay triangulation of \( N \) points in the Euclidean plane can be obtained in \( O(N \log(N)) \) time (see Preparata and Shamos (1985, p. 214)). We used a modification of the Delaunay triangulation algorithm described in Fortune (1987). The code was modified from the implementation available on Netlib (netlib@research.att.com). We then used Kruskal’s algorithm to obtain a minimum spanning tree from the triangulation. Kruskal’s algorithm takes \( O(N \log(N)) \) time, the same as the triangulation.

Let \( S \subseteq \mathcal{S} \) be the set of objects to be clustered, where \( \mathcal{S} \) is a metric space. Let \( Z(s) : \mathcal{S} \rightarrow \mathbb{R}^2 \) be an operator which assigns two real numbers to each element \( s \in S \). To produce a minimum spanning tree of \( S \) one needs a spanning graph \( G \subseteq S \times S \), a distance function \( d(g), g \in G \), and a minimum spanning tree algorithm which, given the graph \( G \) with edge lengths computed using distance function \( d \), produces the minimum spanning tree. Note, that according to our definition, the algorithm is not just the choice of method (such as Prim’s or Kruskal’s) but also the order of the input (vertices for Prim’s algorithm and edges for Kruskal’s). The reason for this is that our data have limited accuracy and thus many ties can occur.

To produce a faster approximate algorithm we could use only a spanning subset of edges instead of the complete graph on \( S \times S \). Alternatively, we could use a simpler distance function, or we could use a combination of these. Using Euclidean distance on the range of the operator \( Z \) we can produce a Delaunay triangulation of \( Z(S) \). Given the triangulation, we need not use Euclidean distance on \( Z(S) \) to produce a minimal spanning tree. We could assign those edges any weight we like and then obtain the minimum spanning tree using those weights.

If we have several operators \( Z_1, \ldots, Z_k \) we can produce \( k \) triangulations and a minimum spanning tree using the spanning subset of edges from the union of those \( k \) triangulations. Note, that the union of edges from \( k \) triangulations cannot have more than \( 3kN \) edges so we can use Kruskal’s algorithm to obtain the minimum spanning tree in \( O(N \log(N)) \) time. There do exist algorithms with running time which is linear in the number of edges, but we felt that Kruskal’s algorithm was adequately fast for this problem.

We would want the spanning tree produced by an approximate algorithm to be close to the one produced by the exact algorithm. To do that we will compare minimum spanning trees produced by various algorithms.
4.1. The choice of metrics

In our data set the distance between the flight legs is a distance between two curves in the three-dimensional space (we chose to not include time). Hence, it might be reasonable to use the Hausdorff distance between the appropriate curves as a distance measure between the flight legs. Let \( s_1 \) and \( s_2 \) be two curves in \( \mathbb{R}^3 \). The Hausdorff metric between two curves is given by

\[
d_H(s_1, s_2) = \max \left( \sup_{x \in s_1} \inf_{y \in s_2} d(x, y), \sup_{y \in s_2} \inf_{x \in s_1} d(x, y) \right),
\]

where \( d(\cdot, \cdot) \) is the Euclidean metric in \( \mathbb{R}^3 \). Since any compact rectifiable curve \( s \) can be parametrized as \( s : I \to \mathbb{R}^3 \) where \( I = [0, 1] \), we can rewrite this distance as

\[
d_H(s_1, s_2) = \max \left( \sup_{t \in I} \inf_{p \in I} d(s_1(t), s_2(p)), \sup_{p \in I} \inf_{t \in I} d(s_1(t), s_2(p)) \right).
\]

The exact computation of the Hausdorff metric involves optimization of possibly non-convex functions. Instead of doing this numerically, we will use the following approximation to the Hausdorff metric. Although it is natural to think of each flight path as a smooth curve, we actually know only several points on it. Hence, each observation is rather considered as a piecewise linear curve consisting of a finite number of line segments. For each curve \( s_i \), let \( I_i = \{t_0, t_1, \ldots, t_n\} \) be the finite subset of \( I \) which corresponds to the endpoints of its component line segments where \( 0 = t_0 \leq t_1 \leq \cdots \leq t_n = 1 \). Then, the approximate Hausdorff metric is computed as

\[
d_{AH}(s_1, s_2) = \max \left( \max_{t \in I_1} \inf_{p \in I_2} d(s_1(t), s_2(p)), \max_{p \in I_1} \inf_{t \in I_2} d(s_1(t), s_2(p)) \right).
\]

Note that since we have replaced the \( \sup_{t \in I} \) with a maximum over a finite subset \( I \), this approximate Hausdorff distance is less than or equal to the true Hausdorff distance.

For a fixed point \( t \in I \), the distance from \( s_1(t) \) to \( s_2 \) is the minimum among the distances from \( s_1(t) \) to component line segments of \( s_2 \). The Euclidean distance between a point \( l(t) \) on a line \( l : \mathbb{R} \to \mathbb{R}^3 \) and another point off the line is a quadratic function in \( t \). Therefore, finding the shortest distance from a fixed point to a line segment turns out to be the constrained minimization of a quadratic function and has a closed-form solution. Hence, \( d_{AH} \) can be exactly computed in a finite number of steps.

We could lower the dimensionality further by considering only the starting and ending points of the curve, which would embed the flights into \( \mathbb{R}^6 \). Noting that the starting and ending points are usually at a very low altitude only four dimensions would matter. Thus, we let the squared symmetrized Euclidean distance between flight legs in \( \mathbb{R}^4 \) be

\[
d_{sf}(s_1, s_2) = \min \left( (x_1^1 - x_2^1)^2 + (y_1^1 - y_2^1)^2 + (x_1^2 - x_2^2)^2 + (y_1^2 - y_2^2)^2, \right.
\]

\[
\times (x_1^1 - x_2^1)^2 + (y_1^1 - y_2^1)^2 + (x_1^2 - x_2^2)^2 + (y_1^2 - y_2^2)^2 \right).
\]
where the lower index indicates flight (1 or 2), the upper index indicates starting or
final point for the flight (s or f), x stands for longitude, and y stands for latitude.

Let \( Z_s(s) \) (or \( Z_f(s) \)) be an operator that gives the latitude and longitude of the
starting (or final) point of flight \( s \). Let \( Z_{sf}(s) = (Z_s(s), Z_f(s)) \). We can find the
triangulation and the minimum spanning tree on the range of \( Z_s \), and \( Z_f \). We can
also find the minimum spanning tree on the union of the edges from both min-
imum spanning trees and/or the union of the edges from both triangulations in
\( N \log(N) \) time (using Kruskal's algorithm). Note that to find the minimum span-
ning tree on the union of edges we can use any metric we like and, in particular,
\( d_{AH} \) or \( d_{sf} \).

5. Application to the data sets

We obtain minimum spanning trees using the various methods, then summarize
and compare the results. We produce plots describing various properties of the trees
we obtained.

5.1. The resulting trees

To make the presentation more concise we will use the following notation. Let
\( D(S, Z) \rightarrow E \) be a Delaunay triangulation operator, where \( S \) is a set of objects in
the space \( \mathcal{S} \) (the space of flight legs), \( Z(s) \rightarrow (x, y) \) operates on elements of \( \mathcal{S} \)
and produces an element of \( \mathbb{R}^2 \), and \( E \) is a set of edges \( (s_i, s_j) \) of the Delaunay
triangulation of the set \( Z(S) \).

Let \( T(E, d) \rightarrow E_1 \) be a minimum spanning tree operator, where \( E \) is a set of edges
\( (s_i, s_j) : s_i, s_j \in \mathcal{S} \) forming a connected graph, \( d((s_i, s_j)) \rightarrow x \) is the distance that gives
a length for the edge \( (s_i, s_j) \), and \( E_1 \) is a minimum spanning tree for the graph \( E \)
with the lengths of edges obtained using distance \( d \).

Remember that \( S \) is the set of all flight legs, \( d_{AH} \) is the approximate Hausdorff
distance, and \( d_{sf} \) is the symmetrized Euclidean distance on the range of \( Z_{sf} \). We
obtained the following minimum spanning trees.

1. \( T_1 = T(S \times S, d_{AH}) \) using Prim's algorithm. This is the MST of the complete
   graph based on the approximate Hausdorff distance.

2. \( T_2 = T(S \times S, d_{sf}) \) using Prim's algorithm. This is the MST of the complete
   graph based on the symmetrized Euclidean distance on the range of \( Z_{sf} \). We
   obtained the following minimum spanning trees.

3. \( T_3 = T(D(S, Z_s), d_{sf}) \) using Kruskal's algorithm. This is the MST based on
   the symmetrized Euclidean distance on the starting and ending points of a flight leg.
   The input edges are from the Delaunay triangulation which is based on the
two-dimensional Euclidean distance on the starting points of a flight leg.

4. \( T_4 = T(D(S, Z_f), d_{sf}) \) using Kruskal's algorithm. This is the MST based on the
   symmetrized Euclidean distance. The input edges are from the Delaunay triangulation
   which is based on the two-dimensional Euclidean distance on the final points of a
   flight leg.
5. $T_5 = T(T_3 \cup T_4, d_{sf})$ using Kruskal’s algorithm. This is the MST based on the symmetrized Euclidean distance. The input edges are the union of the edges in the two MSTs, $T_3$ and $T_4$.

6. $T_6 = T(D(S, Z_s) \cup D(S, Z_{sf}), d_{sf})$ using Kruskal’s algorithm. This is the MST based on the symmetrized Euclidean distance. The input edges are the union of the edges from the two Delaunay triangulations used as input to $T_3$ and $T_4$.

7. $T_7 = T(D(S, Z_s), d_{AH})$ using Kruskal’s algorithm. This is the MST based on the approximate Hausdorff distance. The input edges are from the Delaunay triangulation which is based on the two-dimensional Euclidean distance on the starting points of a flight leg.

8. $T_8 = T(D(S, Z_f), d_{AH})$ using Kruskal’s algorithm. This is the MST based on the approximate Hausdorff distance. The input edges are from the Delaunay triangulation which is based on the two-dimensional Euclidean distance on the final points of a flight leg.

9. $T_9 = T(T_7 \cup T_8, d_{AH})$ using Kruskal’s algorithm. This is the MST based on the approximate Hausdorff distance. The input edges are the union of the edges in the two MSTs, $T_7$ and $T_8$.

In each case where the MST is derived from an incomplete graph it is important to remember that the distance used to select the edges in the incomplete graph need not be (and, in fact, is not) the same as the distance used to construct the MST from the edges in the incomplete graph.

Tables 1, 2, and Fig. 1 contain summary information for each tree. Table 1 contains the total length of each tree (the sum of the lengths of the edges) using two distance functions and the computation time in hours, minutes, and seconds on our Hewlett Packard 9000/735 workstation. Table 2 shows the vertex degree distribution for the trees. It shows the number of vertices (flight legs) that have the specified number of adjacent edges in the tree (1 to 6) and also the maximal degree over all vertices. For comparison, from geometric considerations we know that the maximal vertex degree of an MST in $\mathbb{R}^2$ is 5.

Fig. 1 shows the edge length distribution for the various trees. The left plot shows lengths obtained using distance $d_{AH}$ and the right plot uses distance $d_{sf}$.

<table>
<thead>
<tr>
<th>Table 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary of the trees</td>
</tr>
<tr>
<td>Name</td>
</tr>
<tr>
<td>------</td>
</tr>
<tr>
<td>$T_1$</td>
</tr>
<tr>
<td>$T_2$</td>
</tr>
<tr>
<td>$T_3$</td>
</tr>
<tr>
<td>$T_4$</td>
</tr>
<tr>
<td>$T_5$</td>
</tr>
<tr>
<td>$T_6$</td>
</tr>
<tr>
<td>$T_7$</td>
</tr>
<tr>
<td>$T_8$</td>
</tr>
<tr>
<td>$T_9$</td>
</tr>
</tbody>
</table>
Table 2
Summary of the distributions of the node degrees

<table>
<thead>
<tr>
<th>Name</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>&gt; 6</th>
<th>max</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_1$</td>
<td>21 239</td>
<td>9 461</td>
<td>5 083</td>
<td>2 388</td>
<td>1 222</td>
<td>630</td>
<td>781</td>
<td>26</td>
</tr>
<tr>
<td>$T_2$</td>
<td>11 911</td>
<td>18 701</td>
<td>8 588</td>
<td>1 495</td>
<td>1 055</td>
<td>49</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>$T_3$</td>
<td>14 370</td>
<td>15 579</td>
<td>7 971</td>
<td>2 346</td>
<td>4 54</td>
<td>798</td>
<td>2 385</td>
<td>8</td>
</tr>
<tr>
<td>$T_4$</td>
<td>14 287</td>
<td>15 739</td>
<td>7 890</td>
<td>2 356</td>
<td>4 59</td>
<td>639</td>
<td>1 085</td>
<td>8</td>
</tr>
<tr>
<td>$T_5$</td>
<td>14 806</td>
<td>15 017</td>
<td>7 877</td>
<td>2 490</td>
<td>5 22</td>
<td>814</td>
<td>1 116</td>
<td>8</td>
</tr>
<tr>
<td>$T_6$</td>
<td>14 803</td>
<td>15 020</td>
<td>7 881</td>
<td>2 485</td>
<td>5 23</td>
<td>814</td>
<td>1 116</td>
<td>8</td>
</tr>
<tr>
<td>$T_7$</td>
<td>15 630</td>
<td>14 278</td>
<td>7 302</td>
<td>2 686</td>
<td>726</td>
<td>1 45</td>
<td>3 72</td>
<td>11</td>
</tr>
<tr>
<td>$T_8$</td>
<td>15 552</td>
<td>14 228</td>
<td>7 595</td>
<td>2 584</td>
<td>645</td>
<td>1 60</td>
<td>4 01</td>
<td>10</td>
</tr>
<tr>
<td>$T_9$</td>
<td>16 891</td>
<td>12 933</td>
<td>6 822</td>
<td>2 867</td>
<td>929</td>
<td>292</td>
<td>70</td>
<td>11</td>
</tr>
</tbody>
</table>

Distributions of the log of edge lengths using $d_{AH}$

Distributions of the log of edge lengths using $d_{sf}$

Fig. 1. Distribution of edge lengths using distance $d_{AH}$ and distance $d_{sf}$.

To obtain the triangulation of starting and ending points we had to take care of the points mapped into the same location by $Z$ (the latitude and longitude were reported as hundredths of degrees), as the triangulation algorithm we used was not able to deal with repeated points. We divided the points into equivalence classes (of equal points) and then did triangulation on those classes. There were 35 889 distinct starting points and 34 818 distinct final points. Then to the Delaunay triangulation edges we added all edges between the points in the equivalence class and edges
between all points in two adjacent (according to the triangulation) classes. This set of edges was the one produced by the operator $D$.

Some indication of the data can be gained by inspecting the plots of the starting points of the flight legs and the ending points of the flight legs; see Figs. 2 and 3. Locations of 50 major airports are indicated by circles.

Fig. 2. Starting points.

Fig. 3. Final points.
5.2. Clustering

Because of the nature of these data (one-dimensional manifolds in three-dimensional Euclidean space), it is difficult to display clusters visually. To help the reader understand the nature of the MSTs we have created, we have plotted the first 500 points which entered the MST for T1 in Fig. 4 and for T2 in Fig. 5. These plots are in the form of a "scatterplot matrix" with three plots (altitude versus longitude at the top, latitude versus longitude at the bottom left, and latitude versus altitude at the bottom right). The "points" in the plots are the projection of the flight legs into each two-dimensional subspace.

It is customary to define clusters from a tree by removing all of the long edges and declaring each of the remaining fragments to be a cluster. Because of the size of our data set, because of the dimension of the observation space, and because our primary interest was in selecting observations to inspect, we felt this was not a reasonable approach. Instead, we thought it more reasonable to look at those points which had many edges incident to them. We imagine that these "stars" or "crossroads" would be more representative of the data. Fig. 6 is a plot of the vertex with the highest node degree in T1 and 26 vertices adjacent to it. Fig. 7 is a plot of another vertex in T1 with its 23 neighbours.

Fig. 4. First 500 points in T1.
Fig. 5. First 500 points in T2.

Fig. 6. The star with 26 adjacent vertices.
5.3. Comparing MSTs

Having several approximate algorithms we wanted to compare their performance not only in terms of computer time but also in terms of the tree quality. To do that we compared edges from different trees.

We intended to evaluate the following two approximations:

How T1 is approximated by other trees?

How T2 is approximated by other trees?

Table 3 shows the number of common edges between T1 and other trees. We see that the best approximation in terms of common edges is T9. T9 also is faster to compute and is shorter in terms of $d_{AH}$ (see Table 1).

We were also interested in approximating T2. There are several ties with respect to the distance $d_{sf}$ (i.e., there exist $s_1, s_2 \in S$ such that $s_1 \neq s_2$ and $d_{sf}(s_1, s_2) = 0$). To find the edges equivalent with respect to the distance $d_{sf}$ we took the distance between the edges to be

$$d(e_1, e_2) = \min(d_{sf}(s_1^1, s_2^1), d_{sf}(s_1^2, s_2^2), d_{sf}(s_1^1, s_2^2) + d_{sf}(s_1^2, s_2^1)),$$

where the upper index indicates the end of an edge, and the lower index indicates the edge itself.

Comparing two trees $A$ and $B$, for each edge in tree $A$ we found a closest edge in tree $B$. The edges with distance zero apart we call “equivalent” edges. The percentage of equivalent edges could be thought of as a measure of closeness between two trees.
Table 3
Summary of the edges common with T1

<table>
<thead>
<tr>
<th>First tree</th>
<th>Second tree</th>
<th>Number of common edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_2$</td>
<td>$T_1$</td>
<td>3242</td>
</tr>
<tr>
<td>$T_3$</td>
<td>$T_1$</td>
<td>1536</td>
</tr>
<tr>
<td>$T_4$</td>
<td>$T_1$</td>
<td>1525</td>
</tr>
<tr>
<td>$T_5$</td>
<td>$T_1$</td>
<td>2318</td>
</tr>
<tr>
<td>$T_6$</td>
<td>$T_1$</td>
<td>2318</td>
</tr>
<tr>
<td>$T_7$</td>
<td>$T_1$</td>
<td>1944</td>
</tr>
<tr>
<td>$T_8$</td>
<td>$T_1$</td>
<td>1945</td>
</tr>
<tr>
<td>$T_9$</td>
<td>$T_1$</td>
<td>3526</td>
</tr>
</tbody>
</table>

Table 4
Summary of the "equivalent" edges

<table>
<thead>
<tr>
<th>First tree</th>
<th>Second tree</th>
<th>Number of equivalent edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_1$</td>
<td>$T_2$</td>
<td>3629</td>
</tr>
<tr>
<td>$T_3$</td>
<td>$T_2$</td>
<td>7187</td>
</tr>
<tr>
<td>$T_4$</td>
<td>$T_2$</td>
<td>7540</td>
</tr>
<tr>
<td>$T_5$</td>
<td>$T_2$</td>
<td>12528</td>
</tr>
<tr>
<td>$T_6$</td>
<td>$T_5$</td>
<td>21650</td>
</tr>
<tr>
<td>$T_4$</td>
<td>$T_5$</td>
<td>21453</td>
</tr>
<tr>
<td>$T_6$</td>
<td>$T_5$</td>
<td>40792</td>
</tr>
</tbody>
</table>

Table 4 shows the number of matching edges for several tree pairs. The trees $T_5$ and $T_6$ are almost the same. The trees $T_2$ and $T_5$ have more than 30% of equivalent edges.

We looked at the distribution of lengths of the equivalent edges vs. the distribution for the remaining edges. The distribution of the equivalent edges in $T_1$ and $T_2$ is shown in Fig. 8.

We also looked at the joint distribution of lengths of edges and the distances to the edge on the other tree. In Fig. 9 the x-axis is the length of an edge in tree $T_1$ (using $d_{st}$). The y-axis shows the distance from an edge in $T_1$ to the closest edge on the other tree. The straight line is the line given by the equation $x = y$. The edges equivalent to $T_1$ and $T_2$ are on the line $y = 0$. Most of the differing edges are located near the line $x = y$. The distance function between the edges we used would give $d(e_1, e_2) \approx \text{length}(e_1)$ if the edge $e_2$ is short and is located close to one of the endpoints of $e_1$. 
Fig. 8. Distribution of lengths for equivalent and differing edges in T1 and T2 using distance $d_{ef}$.

Fig. 9. Joint distribution of edge lengths in T1 vs. the distance to the closest edge in T2.
6. Discussion

We have tried to develop a method which aids us in the inspection of a very large high-dimensional data set. In our particular example, the observations are not even vectors, but rather one-dimensional manifolds in $\mathbb{R}^3$.

The basic idea we have used is to construct a minimum spanning tree in a low-dimensional Euclidean space as an approximation to the actual tree. The efficacy of the method is demonstrated by the fact that in our particular problem, construction of the actual minimum spanning tree is essentially infeasible while the construction of approximation requires only a few seconds on a fast workstation (or a few minutes for the more accurate approximation).

It is difficult to measure the accuracy of an approximation in the space of trees simply because of its discrete nature. We chose to use the length of the tree, the edge length distribution of the tree, and the fraction of equivalent edges as our measures of distance. The accuracy of our approximations can be judged by studying the lengths in Table 1, the edge length distributions in Fig. 1, and the equivalent edges in Tables 3 and 4.

We conclude that $T_2$ does not do a very good job of approximating $T_1$ and it takes the longest time to compute as compared to our other approximation methods. We further conclude that $T_3$ and $T_4$ are not very good approximations of either $T_1$ or $T_2$. The trees $T_5$ and $T_6$, improve the approximation to $T_1$, but $T_9$ is substantially better than any other approximation we tried. As $T_9$ takes longer to compute, $T_5$ or $T_6$ are reasonable competitors.

The approximation of $T_2$ by $T_5$ is very good (32% of the edges were equivalent). The trees $T_5$ and $T_6$ were almost identical, but $T_5$ is simpler to compute.

On future data sets we intend to use a method like $T_5$ or $T_9$. The idea behind $T_5$ or $T_9$ is to take the union of the edges from some small number of MSTs and find the MSTs of this set of edges. For $T_5$ and $T_9$ we only used two trees but could have used more.

References

Sedgewick, R., Algorithms (Addison-Wesley, Reading, MA, 1983).