Minimum spanning trees and clusters

Thomas Lumley

January 19, 2010

The Euclidean minimum spanning tree for a set of points is the shortest tree connecting all the points. It can be used for clustering, by dropping the longest edges.

There are three related greedy algorithms for the minimum spanning tree, all of which come very close to the lower bound for asymptotic complexity. Kruskal's algorithm (also known as 'single linkage clustering') adds the shortest edge that does not form a cycle, and ends up with a tree only at the last step. Prim's algorithm maintains a tree at all times, adding the shortest edge that connects a point in the tree to a point outside the tree. Boruvka's algorithm maintains a forest that is reduced at each step by connecting each tree to its closest neighbour.

Kruskal's algorithm is efficient for sparse graphs but requires $O(n^2)$ space for a Euclidean spanning tree on n points. Boruvka's algorithm is attractive largely because it can be run in parallel very efficiently. Prim's algorithm is easiest to implement for large Euclidean minimum spanning trees.

For general graphs or for very high dimensional spaces, an efficient way to implement Prim's algorithm is for each point outside the tree to keep track of its nearest neighbour in the tree. These potential links are then placed in a priority queue. The algorithms are described in detail in Sedgewick's *Algorithms*.

For moderate-dimensional Euclidean space there is more efficient strategy, relying on the ability to find nearest neighbours efficiently using a k-d tree. Each point in the tree keeps track of its nearest neighbour outside the tree. These potential links are placed in a priority queue ordered by length, and the shortest link is removed and used at each step. Lazy updating is used: when a link reaches the front of the queue, its target point may already have been added, in which case it is updated with the new nearest neighbour outside the tree and replaced in the queue.

If the data are in the form of tight, well-separated clusters, the nearest-neighbour finder will be highly efficient towards the center of the cluster, but slower when dealing with outliers and debris. Combining features of Boruvka's and Prim's algorithms we can run Prim's algorithm until the next link to be added is longer than a threshold, then restart the tree at a randomly chosen point. The result is likely to be a small forest of large trees representing the distinct clusters, with an additional small tree representing the isolated points. Approaches similar to this were first developed by Bentley & Friedman (1978), but their 'multifragment' algorithm focused on constructing the full tree, rather than on finding clusters. The restarting approach appears to perform very well when the data in fact consist of well-separated clusters. It does not make assumptions about the shape of the clusters and does not need the number of clusters to be specified in advance, but does require specifying a threshold for nearest-neighbour distance.

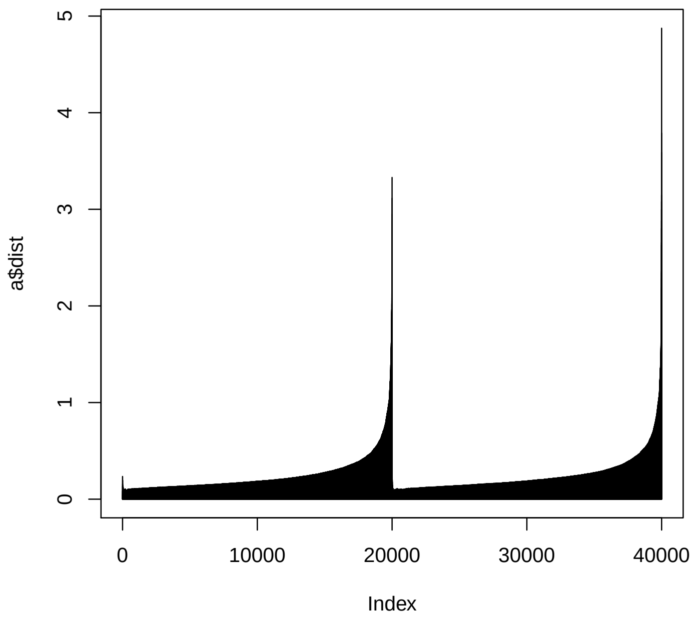
For example, consider 40000 points in five dimensions, sampled from an equal mixture of $N_5(0, 1^2)$ and $N_5(4, 1^2)$.

```
> library(nnclust)
> set.seed(31337)
> x <- matrix(rnorm(1e+05), ncol = 5)
> x <- rbind(x + 4, matrix(rnorm(1e+05), ncol = 5))
> system.time(a <- mst(x))
```

```
   user  system elapsed
 16.232   0.059  16.559
```

Prim's algorithm fills one cluster before crossing over to the next, as is shown by a plot of the successive distances. Because of the difficulty in moving between clusters it runs more slowly than it would with two clusters from, say, $N(0, 1)$ and $N(2, 1)$.

```
> plot(a$dist, type = "h")
```



Setting a threshold of 0.5 and restarting the tree gives two trees containing more than 90% of the points, in less than 20% of the time. If necessary, the isolated points can be tidied into a third tree in very little extra time

```
> system.time({
+     b <- mst_restart(x, threshold = 0.5)
+     d <- mst_restart(x[-b$to, ], threshold = 0.5)
+ })
```

```
   user  system elapsed 
  3.036   0.031   3.176 
```

```
> print((d$n + b$n)/nrow(x))
```

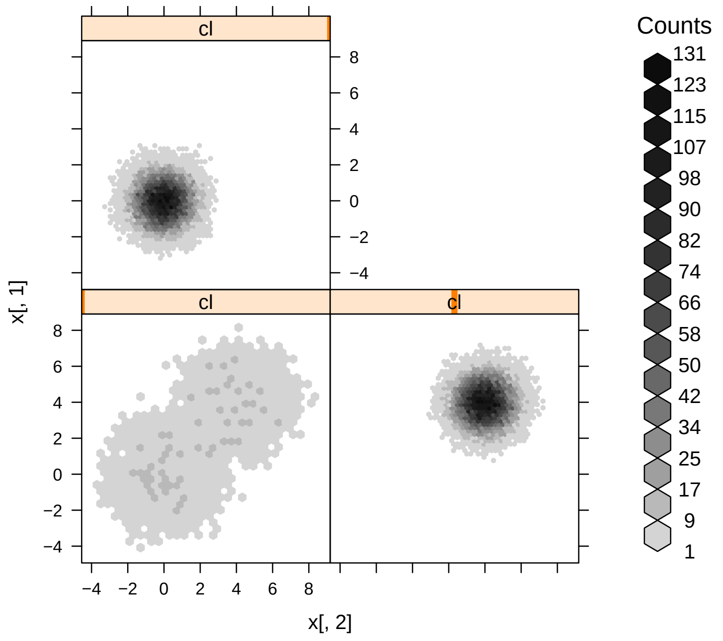
```
[1] 0.928125
```

```
> system.time(f <- mst(x[-b$to, ][-d$to, ]))
```

```
   user  system elapsed 
  0.185   0.002   0.195 
```

A hexbin plot of the two trees and the remaining points shows that the clusters have been identified correctly.

```
> library(hexbin)
> cl <- rep(1, nrow(x))
> cl[b$to] <- 2
> cl[-b$to][d$to] <- 3
> show(hexbinplot(x[, 1] ~ x[, 2] | cl))
```



Rather than restarting at a random point it is better to find nearest neighbours in the remaining set of points and restart at the point with the closest neighbour, which is likely to be near the center of a cluster. This is done by `nncluster`

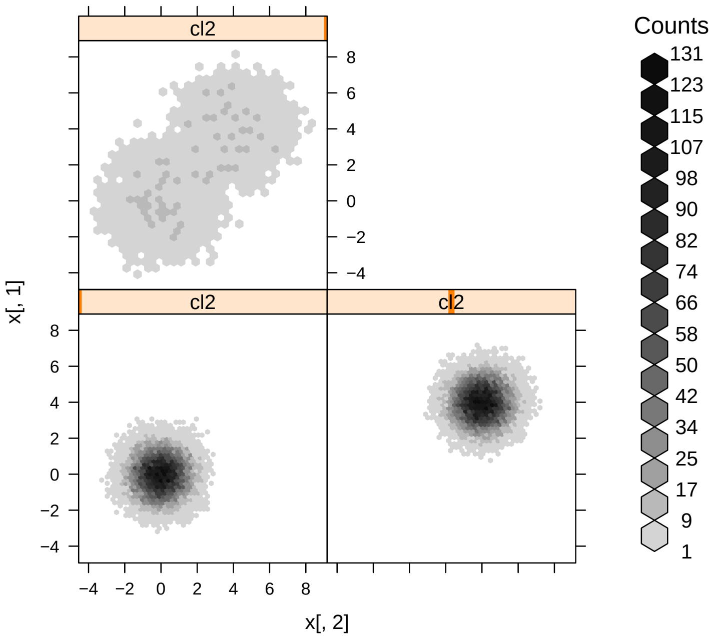
```
> system.time(forest <- nncluster(x, threshold = 0.5))

   user  system elapsed 
  4.027   0.057   4.203 

> forest

MST-based clustering in  5  dimensions
Cluster sizes: 18547 18579 
 and  2874  outliers
Threshold = 0.5 

> cl2 <- clusterMember(forest)
> show(hexbinplot(x[, 1] ~ x[, 2] | cl2))
```



References

1. Bentley JL, Friedman JH (1978) Fast algorithm for constructing minimal spanning trees in coordinate spaces. *IEEE Transactions on Computers* C-27(2).

2. Sedgewick R. (2001) *Algorithms in C: Part 5 Graph Algorithms* 3rd edition. Addison-Wesley.