We can design a greedy algorithm by running sort of a “backward” version of Kruskal’s Algorithm. Specifically, we start with the full graph (*V*, *E*) and begin deleting edges in order of decreasing cost. As we get to each edge *e* (starting from the most expensive), we delete it as long as doing so would not actually disconnect the graph we currently have. For want of a better name, this approach is generally called the Reverse-Delete Algorithm.

1) Assuming the graph is connected, try to prove that Reverse-Delete is a correct algorithm for finding the MST.

The idea is to show that MSTs will not be changed with each deletion. If G is the original graph, let G’ be the graph after one deletion by Reverse-Delete. Now, since the graph is still connected, the edge *e* that was removed must be part of a cycle (if not, deleting it would disconnect the graph… prove this). Since *e* was the heaviest edge available, it is also the heaviest edge of that cycle. We’ll use the following lemma.

> Lemma: Let C be a cycle in a graph G with $e \in C$ the edge with largest weight in the cycle. Then *e* cannot belong to any MST of G.
>
> Proof of lemma: Assume *e* actually is part of some MST (if not we’re done). Removing *e* from the MST disconnects the graph into two components. However, there exists an edge *e’* in the cycle that isn’t used and will connect the graph. Reconnect the graph using that edge *e*’. Now it is a spanning tree with smaller weight since w(*e*’) < w(*e*). Since this contradicts that we started with an MST in the first place, *e* cannot be part of any MST.

So, by the lemma, the edges removed by Reverse-Delete are not part of any MST. Thus the MST of G’ will be the same as an MST of G.

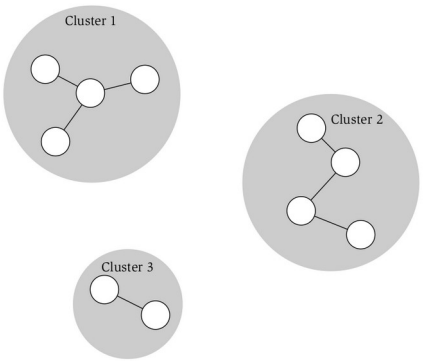
2) List some disadvantages of Reverse-Delete compared to Kruskal's algorithm.

For every edge that we hope to delete, we need to check whether the graph is disconnected. This is a costly check (can be done with a search).

3) ***Clusterings of Maximum Spacing*** Minimum spanning trees play a role in one of the most basic formalizations, which we describe here. Suppose we are given a set $U$ of $n$ objects, labeled $p_1, p_2, \ldots, p_n$. For each pair, $p_i$ and $p_j$, we have a numerical distance $d(p_i, p_j)$. We require only that $d(p_i, p_i) = 0$; that $d(p_i, p_j) > 0$ for distinct $p_i$ and $p_j$; and that distances are symmetric: $d(p_i, p_j) = d(p_j, p_i)$.

Suppose we are seeking to divide the objects in $U$ into $k$ groups, for a given parameter $k$. We say that a *$k$-clustering* of $U$ is a partition of $U$ into $k$ nonempty sets $C_1, C_2, \ldots, C_k$. We define the *spacing* of a $k$-clustering to be the minimum distance between any pair of points lying in different clusters. Given that we want points in different clusters to be far apart from one another, a natural goal is to seek the $k$-clustering with the maximum possible spacing.

The question now becomes the following. There are exponentially many different $k$-clusterings of a set $U$; how can we efficiently find the one that has maximum spacing?




Design a clustering algorithm using spanning tree ideas.

Any minimum spanning tree algorithm that stops at |E| = |V| - k will give the correct clustering.

**(4.26)** *The components $C_1, C_2, \ldots, C_k$ formed by deleting the $k-1$ most expensive edges of the minimum spanning tree $T$ constitute a $k$-clustering of maximum spacing.*

**Proof.** Let $\mathcal{C}$ denote the clustering $C_1, C_2, \ldots, C_k$. The spacing of $\mathcal{C}$ is precisely the length $d^*$ of the $(k-1)^{st}$ most expensive edge in the minimum spanning tree; this is the length of the edge that Kruskal's Algorithm would have added next, at the moment we stopped it.

Now consider some other $k$-clustering $\mathcal{C}'$, which partitions $U$ into nonempty sets $C'_1, C'_2, \ldots, C'_k$. We must show that the spacing of $\mathcal{C}'$ is at most $d^*$.

Since the two clusterings $\mathcal{C}$ and $\mathcal{C}'$ are not the same, it must be that one of our clusters $C_r$ is not a subset of any of the $k$ sets $C'_s$ in $\mathcal{C}'$. Hence there are points $p_i, p_j \in C_r$ that belong to different clusters in $\mathcal{C}'$—say, $p_i \in C'_s$ and $p_j \in C'_t \neq C'_s$.

Now consider the picture in Figure 4.15. Since $p_i$ and $p_j$ belong to the same component $C_r$, it must be that Kruskal's Algorithm added all the edges of a $p_i$-$p_j$ path $P$ before we stopped it. In particular, this means that each edge on

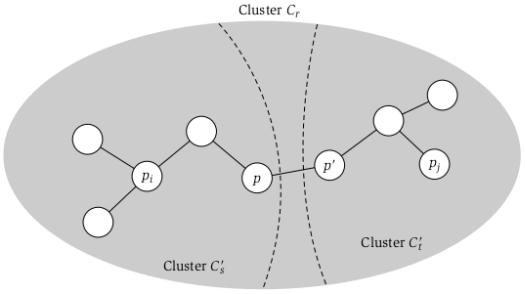



**Figure 4.15** An illustration of the proof of (4.26), showing that the spacing of any other clustering can be no larger than that of the clustering found by the single-linkage algorithm.

$P$ has length at most $d^*$. Now, we know that $p_i \in C'_s$ but $p_j \notin C'_s$; so let $p'$ be the first node on $P$ that does not belong to $C'_s$, and let $p$ be the node on $P$ that comes just before $p'$. We have just argued that $d(p, p') \leq d^*$, since the edge $(p, p')$ was added by Kruskal's Algorithm. But $p$ and $p'$ belong to different sets in the clustering $\mathcal{C}'$, and hence the spacing of $\mathcal{C}'$ is at most $d(p, p') \leq d^*$. This completes the proof. ■