Applications of Minimum Spanning Trees

T. M. Murali

February 11, 2008

Minimum Spanning Trees

- We motivated MSTs through the problem of finding a low-cost network connecting a set of nodes.
- MSTs are useful in a number of seemingly disparate applications.
- We will consider two problems: clustering (Chapter 4.7) and minimum bottleneck graphs (problem 9 in Chapter 4).

Motivation for Clustering

- Given a set of objects and distances between them.
- Objects can be images, web pages, people, species ....
- Distance function: increasing distance corresponds to decreasing similarity.
- Goal: group objects into clusters, where each cluster is a set of similar objects.

Formalising the Clustering Problem

- Let U be the set of n objects labelled $p_1, p_2, \ldots, p_n$.
- For every pair p_i and p_j, we have a distance $d(p_i, p_j)$.
- We require $d(p_i, p_i) = 0$, $d(p_i, p_j) > 0$, if $i \neq j$, and $d(p_i, p_j) = d(p_j, p_i)$

Formalising the Clustering Problem

- Let U be the set of n objects labelled $p_1, p_2, \ldots, p_n$.
- For every pair p_i and p_j, we have a distance $d(p_i, p_j)$.
- We require $d(p_i, p_i) = 0$, $d(p_i, p_j) > 0$, if $i \neq j$, and $d(p_i, p_j) = d(p_j, p_i)$
- Given a positive integer k, a *k-clustering* of U is a partition of U into k non-empty subsets or "clusters" $C_1, C_2, \ldots C_k$.

Formalising the Clustering Problem

- Let U be the set of n objects labelled $p_1, p_2, \ldots, p_n$.
- For every pair p_i and p_j, we have a distance $d(p_i, p_j)$.
- We require $d(p_i, p_i) = 0$, $d(p_i, p_j) > 0$, if $i \neq j$, and $d(p_i, p_j) = d(p_j, p_i)$
- Given a positive integer k, a *k-clustering* of U is a partition of U into k non-empty subsets or "clusters" $C_1, C_2, \ldots C_k$.
- The *spacing* of a clustering is the smallest distance between objects in two different subsets:

$$\text{spacing}(C_1, C_2, \ldots C_k) = \min_{\substack{1 \leq i,j \leq k, i \neq j, \\ p \in C_i, q \in C_j}} d(p, q)$$

Formalising the Clustering Problem

- Let U be the set of n objects labelled $p_1, p_2, \ldots, p_n$.
- For every pair p_i and p_j, we have a distance $d(p_i, p_j)$.
- We require $d(p_i, p_i) = 0$, $d(p_i, p_j) > 0$, if $i \neq j$, and $d(p_i, p_j) = d(p_j, p_i)$
- Given a positive integer k, a *k-clustering* of U is a partition of U into k non-empty subsets or "clusters" $C_1, C_2, \ldots C_k$.
- The *spacing* of a clustering is the smallest distance between objects in two different subsets:

$$\text{spacing}(C_1, C_2, \ldots C_k) = \min_{\substack{1 \leq i,j \leq k, i \neq j, \\ p \in C_i, q \in C_j}} d(p, q)$$

CLUSTERING OF MAXIMUM SPACING

INSTANCE: A set U of objects, a distance function $d : U \times U \rightarrow \mathbb{R}^+$, and a positive integer K

SOLUTION: A k-clustering of U whose spacing is the largest over all possible k-clusterings.

Algorithm for Clustering of Maximum Spacing

- Intuition: greedily cluster objects in increasing order of distance.

Algorithm for Clustering of Maximum Spacing

- Intuition: greedily cluster objects in increasing order of distance.
- Let $\mathcal{C}$ be a set of n clusters, with each object in U in its own cluster.
- Process pairs of objects in increasing order of distance.
  - Let (p, q) be the next pair with $p \in C_p$ and $q \in C_q$.
  - If $C_p \neq C_q$, add new cluster $C_p \cup C_q$ to $\mathcal{C}$, delete C_p and C_q from $\mathcal{C}$.
- Stop when there are k clusters in $\mathcal{C}$.

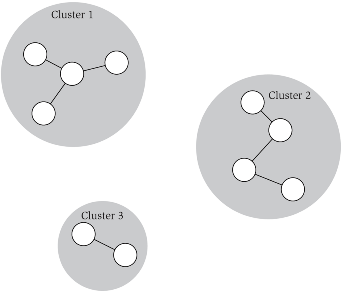


Figure 4.14 An example of single-linkage clustering with $k = 3$ clusters. The clusters are formed by adding edges between points in order of increasing distance.

Algorithm for Clustering of Maximum Spacing

- Intuition: greedily cluster objects in increasing order of distance.
- Let $\mathcal{C}$ be a set of n clusters, with each object in U in its own cluster.
- Process pairs of objects in increasing order of distance.
  - Let (p, q) be the next pair with $p \in C_p$ and $q \in C_q$.
  - If $C_p \neq C_q$, add new cluster $C_p \cup C_q$ to $\mathcal{C}$, delete C_p and C_q from $\mathcal{C}$.
- Stop when there are k clusters in $\mathcal{C}$.
- Same as Kruskal's algorithm but do not add last $k-1$ edges in MST.

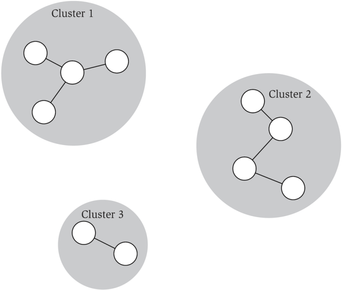


Figure 4.14 An example of single-linkage clustering with $k = 3$ clusters. The clusters are formed by adding edges between points in order of increasing distance.

Why does the Algorithm Work?

- Let $\mathcal{C}$ be the clustering produced by the algorithm and let $\mathcal{C}'$ be any other clustering.
- What is spacing($\mathcal{C}$)?

Why does the Algorithm Work?

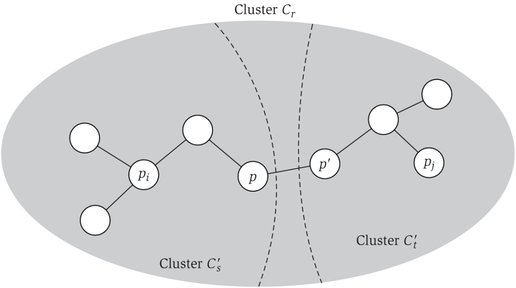
- Let $\mathcal{C}$ be the clustering produced by the algorithm and let $\mathcal{C}'$ be any other clustering.
- What is spacing($\mathcal{C}$)? It is the cost of the $(k-1)$st most expensive edge in the MST. Let this cost be d^*.
- We will prove that spacing($\mathcal{C}'$) $\leq d^*$.

spacing$(\mathcal{C}') \leq d^*$

- There must be two points p_i and p_j in U such that the belong to the same cluster C_r in $\mathcal{C}$ but to different clusters in $\mathcal{C}'$.
- Suppose $p_i \in C'_s$ and $p_j \in C'_t$ in $\mathcal{C}'$.

spacing($\mathcal{C}'$) $\leq d^*$

- There must be two points p_i and p_j in U such that the belong to the same cluster C_r in $\mathcal{C}$ but to different clusters in $\mathcal{C}'$.
- Suppose $p_i \in C'_s$ and $p_j \in C'_t$ in $\mathcal{C}'$.
- All edges in the path connecting p_i and p_j in the MST have length $\leq d^*$.
- In particular, there is a point $p \in C'_s$ and a point $p' \in C'_t$ such that $d(p, p') \leq d* \Rightarrow \text{spacing}(\mathcal{C}') \leq d^*$.



Minimum Bottleneck Spanning Tree (MBST)

- The MST minimises the total cost of a spanning network.
- Consider another network design criterion: compute a spanning tree in which the most expensive edge is as cheap as possible.

Minimum Bottleneck Spanning Tree (MBST)

- The MST minimises the total cost of a spanning network.
- Consider another network design criterion: compute a spanning tree in which the most expensive edge is as cheap as possible.
- In an undirected graph $G(V, E)$, let (V, T) be a spanning tree. The *bottleneck edge* in T is the edge with largest cost in T.

Minimum Bottleneck Spanning Tree (MBST)

- The MST minimises the total cost of a spanning network.
- Consider another network design criterion: compute a spanning tree in which the most expensive edge is as cheap as possible.
- In an undirected graph $G(V, E)$, let (V, T) be a spanning tree. The *bottleneck edge* in T is the edge with largest cost in T.

MINIMUM BOTTLENECK SPANNING TREE (MBST)

INSTANCE: An undirected graph $G(V, E)$ and a function $c : E \rightarrow \mathbb{R}^+$

SOLUTION: A set $T \subseteq E$ of edges such that (V, T) is connected and there is no spanning tree in G with a cheaper bottleneck edge.

Two Questions on MBSTs

1. Assume edge costs are distinct.
2. Is every MBST tree an MST?

3. Is every MST an MBST?

Two Questions on MBSTs

1. Assume edge costs are distinct.
2. Is every MBST tree an MST? No. It is easy to create a counterexample.
3. Is every MST an MBST? Yes. Use the cycle property.
   - Let T be the MST and let T' be a spanning tree with a cheaper bottleneck edge. Let e be the bottleneck edge in T.
   - Every edge in T' is cheaper than e.
   - Adding e to T' creates a cycle consisting only of edges in T' and e.
   - Since e is the costliest edge in this cycle, by the cycle property, e cannot belong to any MST, which contradicts the fact that T is an MST.