CMSC 451: Minimum Spanning Trees & Clustering

Slides By: Carl Kingsford

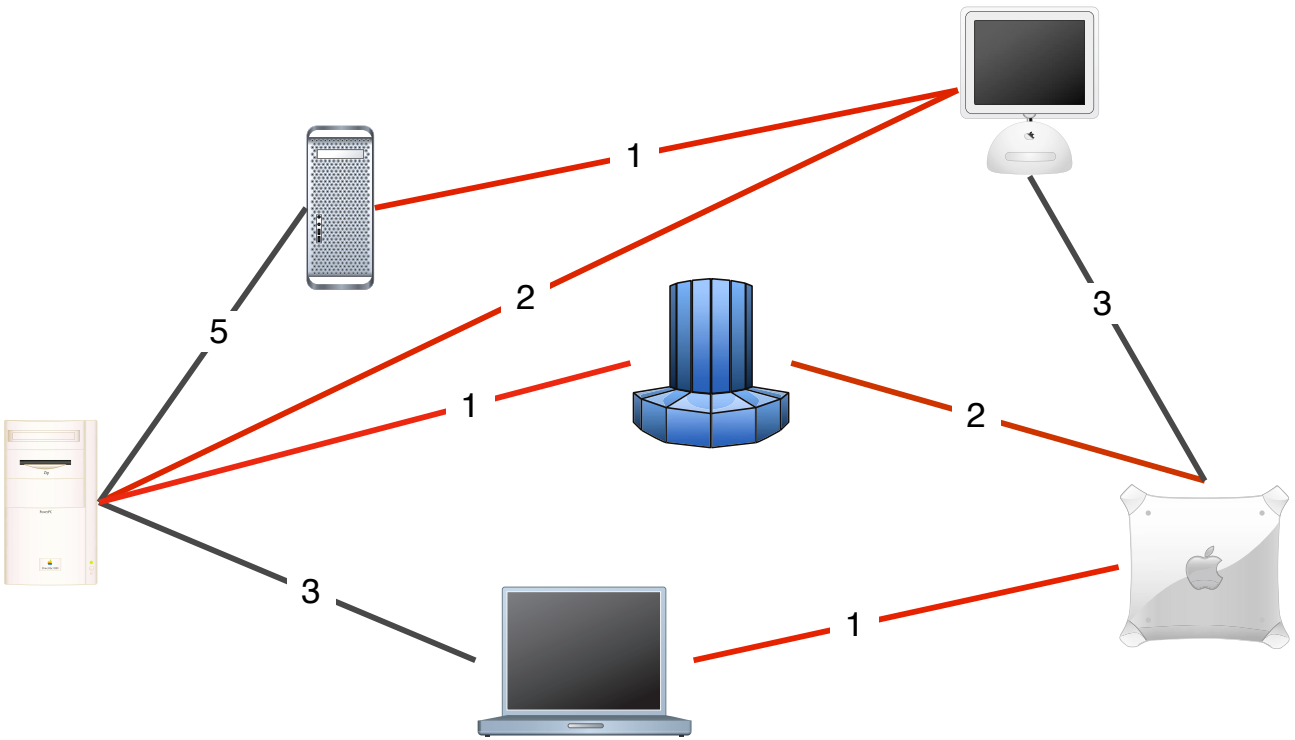
Department of Computer Science
University of Maryland, College Park

Based on Sections 4.5–4.6 of *Algorithm Design* by Kleinberg & Tardos.

Network Design

You want to connect up several computers with a network, and you want to run as little wire as possible.

It is feasible to directly connect only some pairs of computers.



Minimum Spanning Tree Problem

Minimum Spanning Tree Problem

Given

- undirected graph G with vertices for each of n objects
- weights $d(u, v)$ on the edges giving the distance u and v,

Find the subgraph T that connects all vertices and minimizes $\sum_{\{u,v\} \in T} d(u, v)$.

T will be a tree. Why?

Minimum Spanning Tree Problem

Minimum Spanning Tree Problem

Given

- undirected graph G with vertices for each of n objects
- weights $d(u, v)$ on the edges giving the distance u and v,

Find the subgraph T that connects all vertices and minimizes $\sum_{\{u,v\} \in T} d(u, v)$.

T will be a tree. Why?

If there was a cycle, we could remove any edge on the cycle to get a new subgraph T' with smaller $\sum_{\{u,v\} \in T'} d(u, v)$.

MST History

- Studied as far back as 1926 by Borůvka.
- We'll see algorithms that take $O(m \log n)$ time, where m is number of edges.
- Best known algorithm takes time $O(m\alpha(m, n))$, where $\alpha(m, n)$ is the "inverse Ackerman" function (grows *very* slowly).
- Still open: Can you find a $O(m)$ algorithm?

Assumption

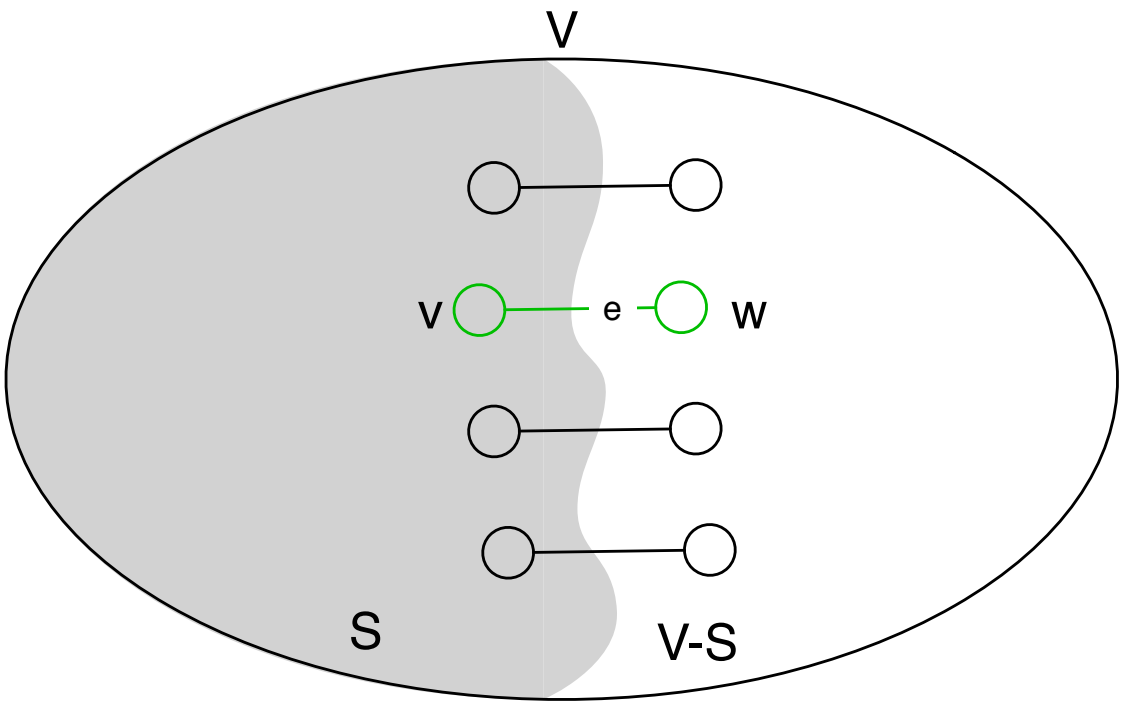
We assume no two edges have the same edge cost.

If this doesn't hold true, we can add a very small value ϵ_e to the weight of every edge e.

Cut Property

Theorem

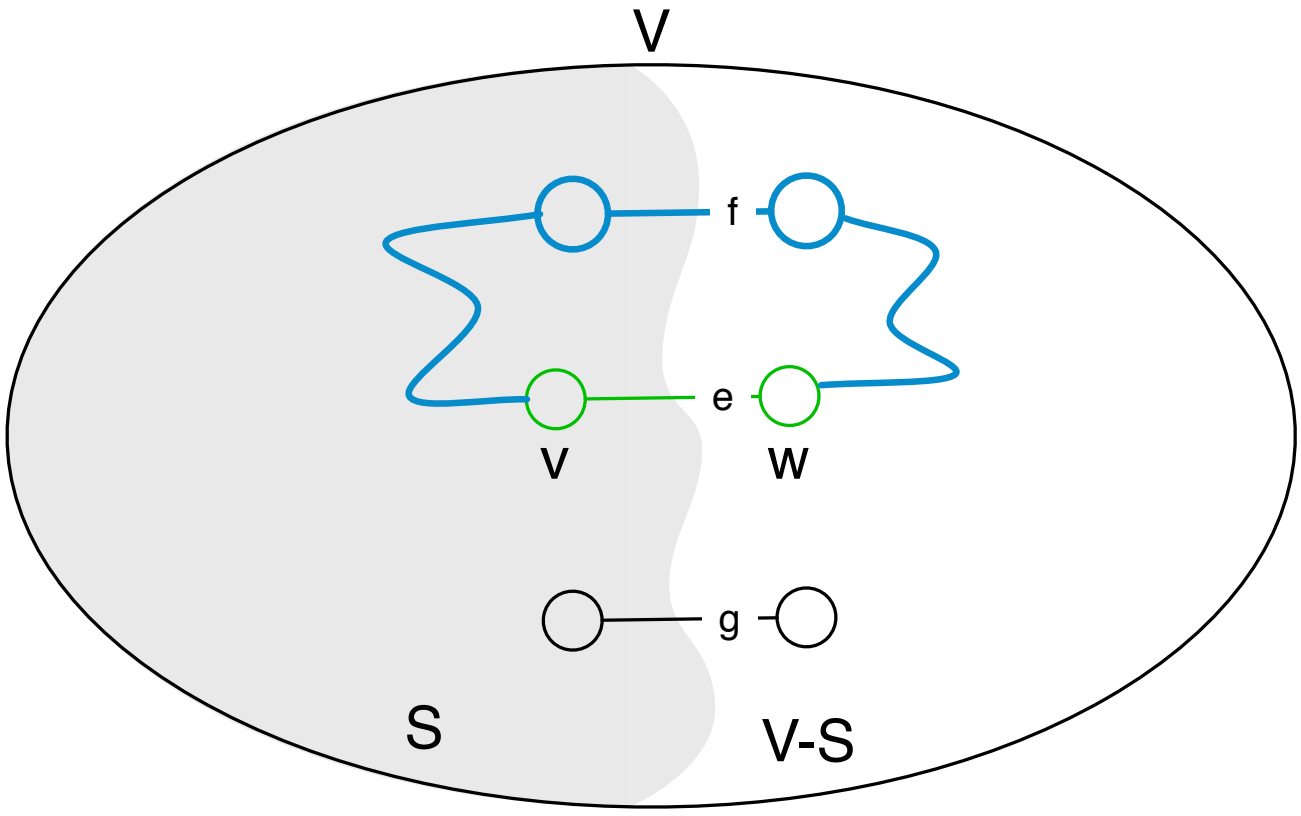
Let S be a subset of nodes, with $|S| \geq 1$ and $|S| \leq n$. Every MST contains the edge $e = (v, w)$ with $v \in S$ and $w \in V - S$ that has minimum weight.



Cut Property, Proof

Suppose T doesn't contain e. Because T is connected, it must contain a path P between v and w. P must contain some edge f that "crosses the cut."

The subgraph $T' = T - f \cup e$ has lower weight than T. T' is acyclic because the only cycle in $T' \cup f$ is eliminated by removing f.



Cycle Property

Theorem (Cycle Property)

Let C be a cycle in G. Let $e = (u, v)$ be the edge with maximum weight on C. Then e is not in any MST of G.

Suppose the theorem is false. Let T be a MST that contains e.

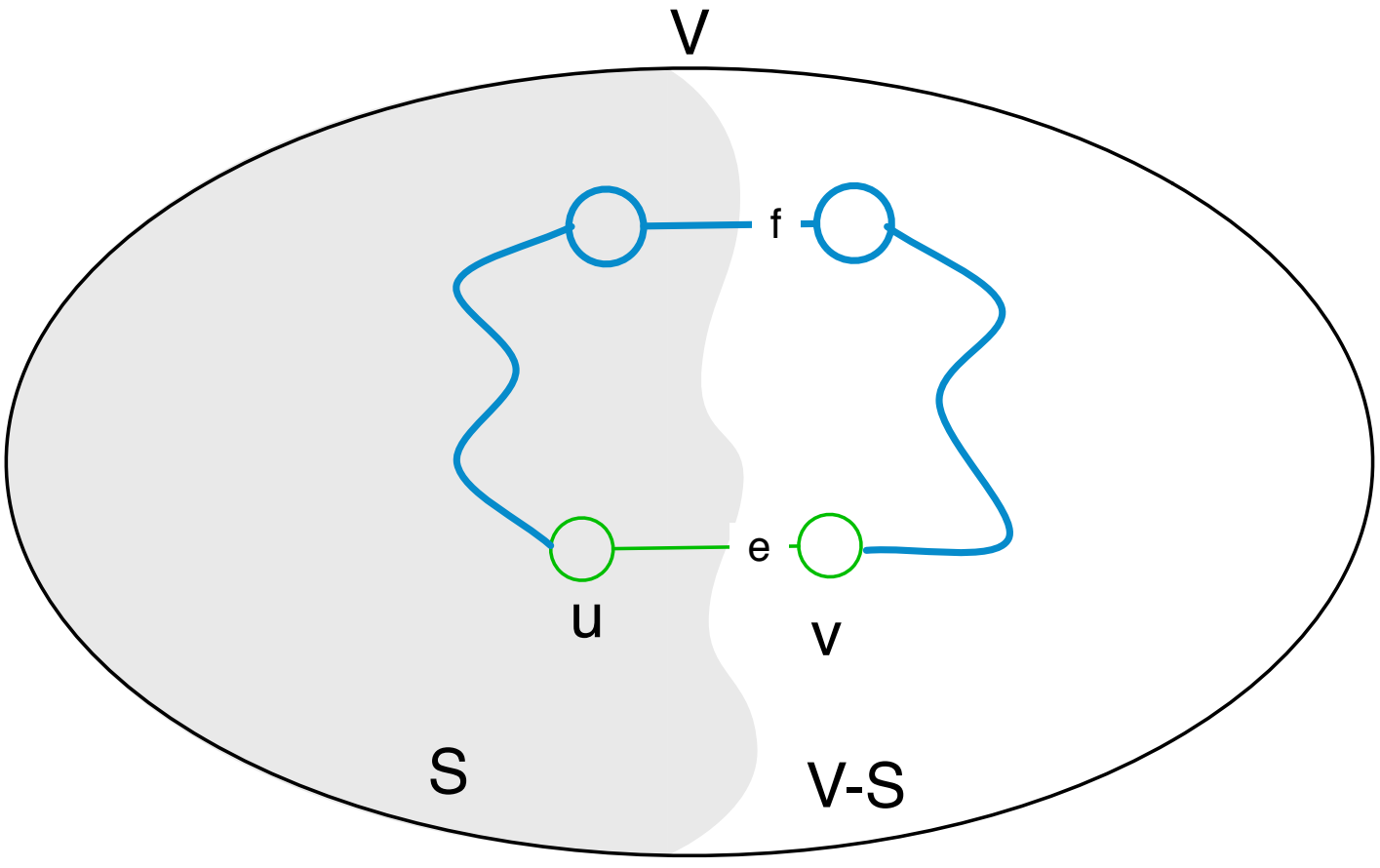
Deleting e from T partitions vertices into 2 sets:

$$S \text{ (that contains } u\text{) and } V - S \text{ (that contains } v\text{)}.$$

Cycle C must have some *other* edge f that goes from S and $V - S$.

Replacing e by f produces a lower cost tree, contradicting that T is an MST.

Cycle Property, Picture



MST Property Summary

1. Cut Property: The smallest edge crossing any cut must be in all MSTs.
2. Cycle Property: The largest edge on any cycle is never in any MST.

Greedy MST Rules

All of these greedy rules work:

1. Add edges in increasing weight, skipping those whose addition would create a cycle. (**Kruskal's Algorithm**)

2. Run `TreeGrowing` starting with any root node, adding the frontier edge with the smallest weight. (**Prim's Algorithm**)

3. Start with all edges, remove them in decreasing order of weight, skipping those whose removal would disconnect the graph. (**"Reverse-Delete" Algorithm**)

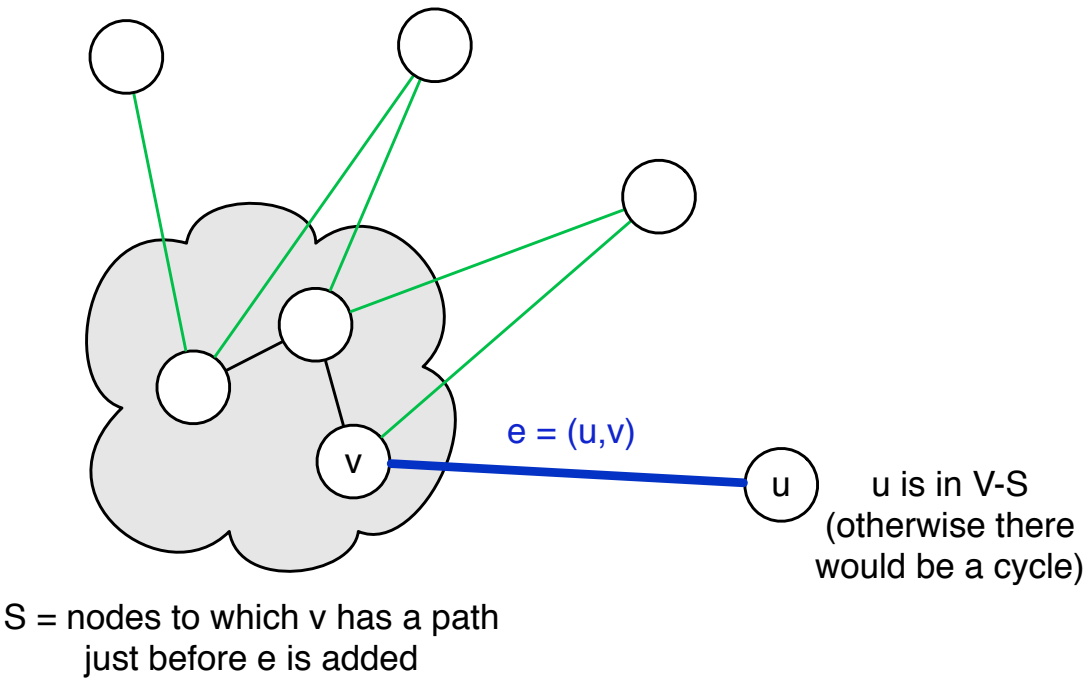
Kruskal's Algorithm

Kruskal's Algorithm: Add edges in increasing weight, skipping those whose addition would create a cycle.

Theorem

Kruskal's algorithm produces a minimum spanning tree.

Proof. Consider the point when edge $e = (u, v)$ is added:

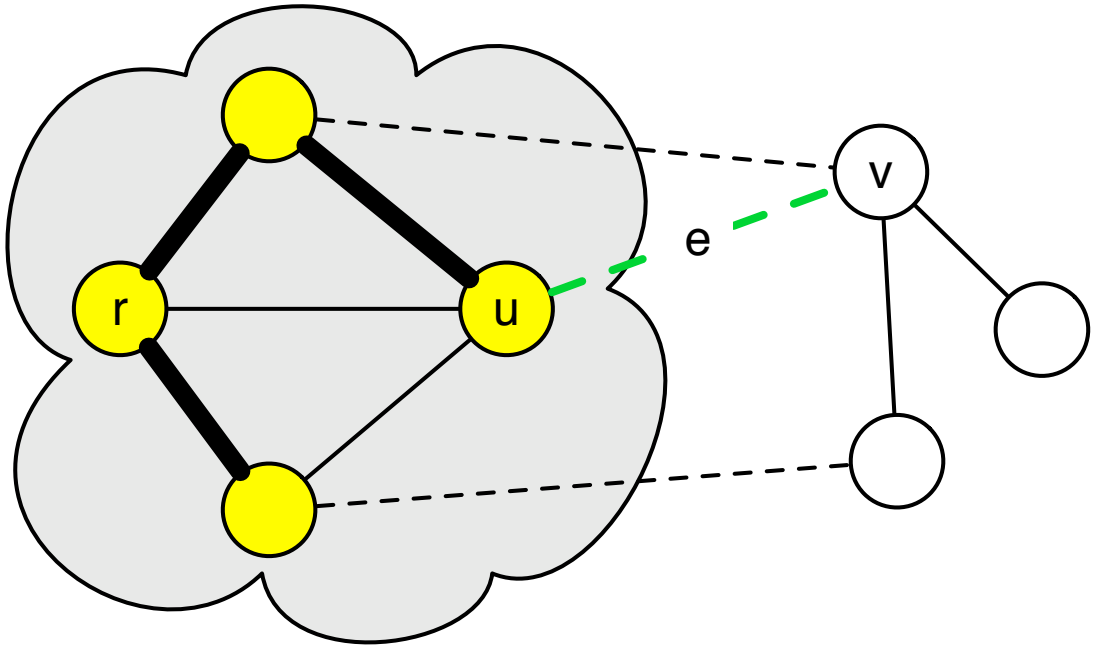


Prim's Algorithm

Prim's Algorithm: Run `TreeGrowing` starting with any root node, adding the frontier edge with the smallest weight.

Theorem

Prim's algorithm produces a minimum spanning tree.



S = set of nodes already in the tree when e is added

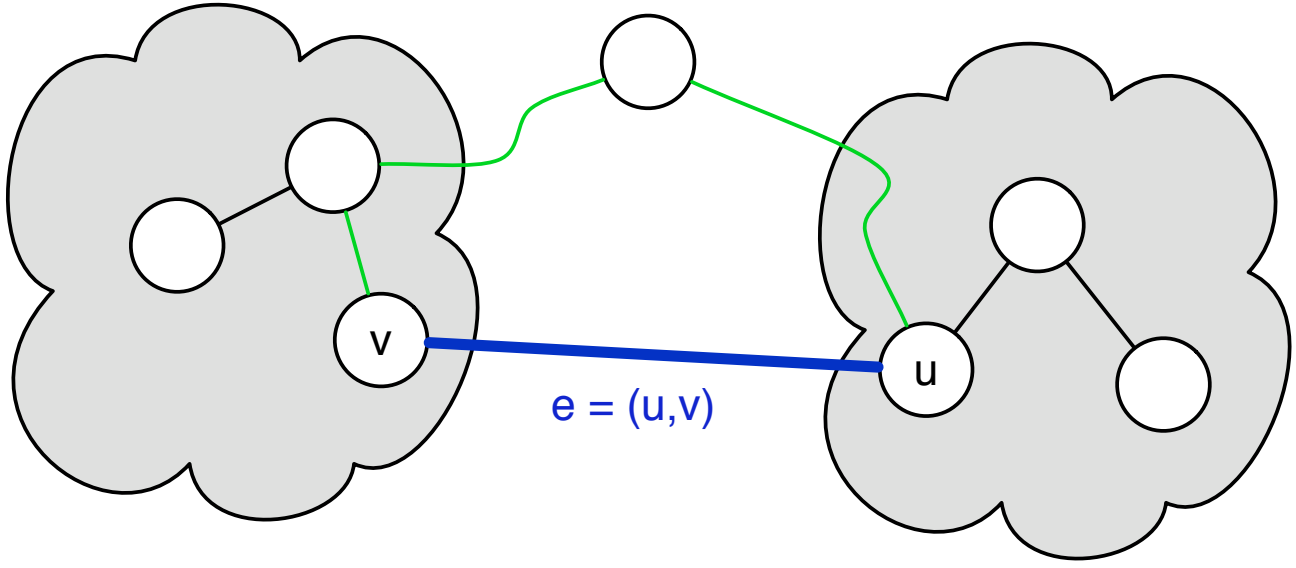
Reverse-Delete Algorithm

Reverse-Delete Algorithm: Remove edges in decreasing order of weight, skipping those whose removal would disconnect the graph.

Theorem

Reverse-Delete algorithm produces a minimum spanning tree.

Because removing e won't disconnect the graph,
there must be another path between u and v



Because we're removing in order of decreasing weight,
e must be the largest edge on that cycle.

Implementation: Prim's & Dijkstra's

- Store the nodes on the frontier in a priority queue, using key:

$$\textbf{Prim's:} \qquad p(v) = \min_{(u,v):u \in S} d(u,v)$$

$$\textbf{Dijkstra's:} \qquad s(v) = \min_{(u,v):u \in S} \mathrm{dist}(s,u) + d(u,v)$$

- `ExtractMin` takes $O(1)$ time, and we do $O(n)$ of them.
- `ChangeMin` takes $O(\log n)$ time, and we do $O(m)$ of them.

Total run time: $O(m \log n)$.

Can implement Kruskal's algorithm in $O(m \log n)$ time too, with more complicated data structures.

Clustering: an application of MST

Clustering

You're given n items and the distance $d(u, v)$ between each of pair.

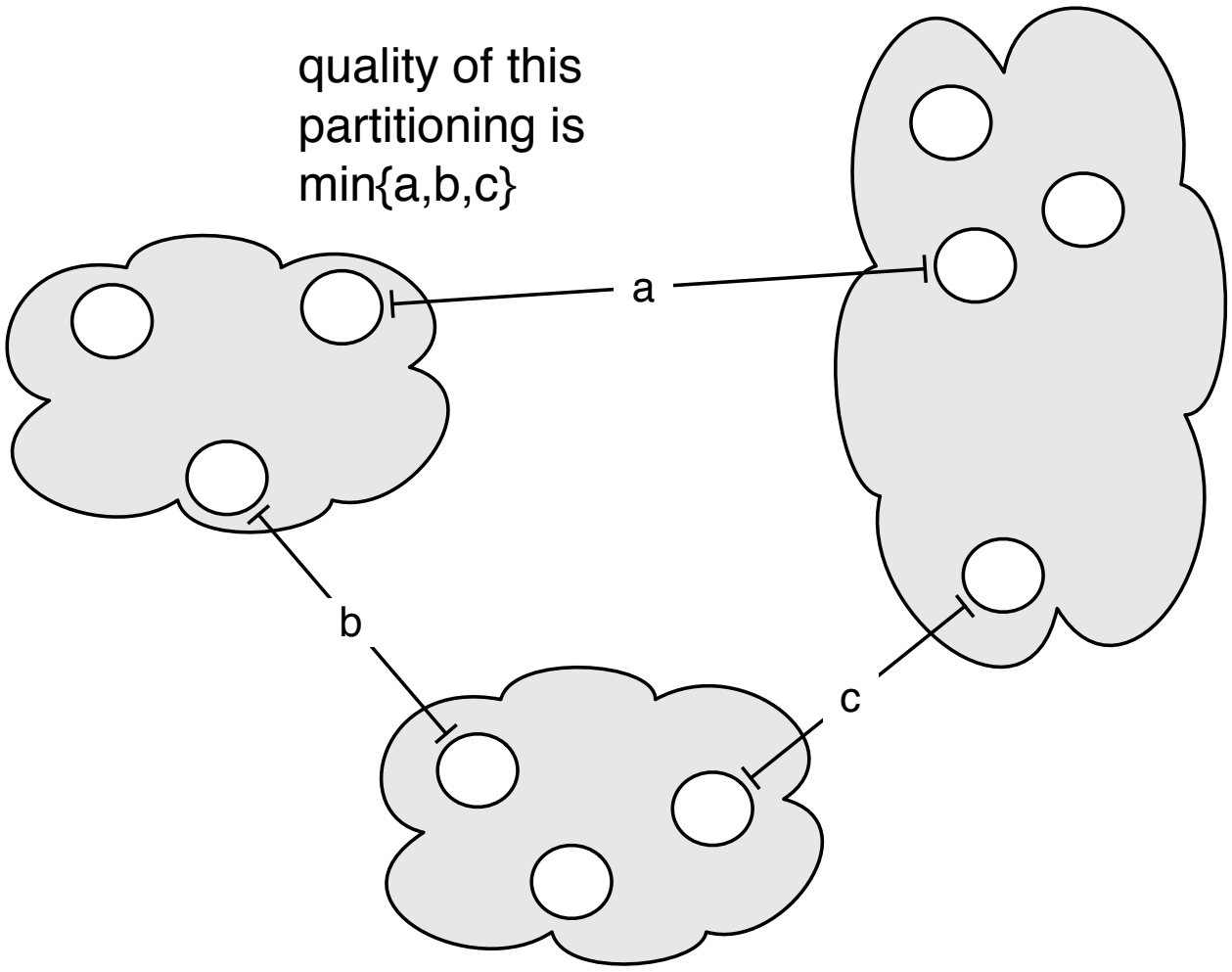
$d(u, v)$ may be an actual distance, or some abstract representation of how dissimilar two things are.

(What's the "distance" between two species?)

Our Goal: Divide the n items up into k groups so that the minimum distance between items in different groups is maximized.

Clustering

Our Goal: Divide the n items up into k groups so that the minimum distance between items in different groups is maximized.



Maximum Minimum Distance

Idea:

- Maintain clusters as a set of connected components of a graph.
- Iteratively combine the clusters containing the two closest items by adding an edge between them.
- Stop when there are k clusters.

Maximum Minimum Distance

Idea:

- Maintain clusters as a set of connected components of a graph.
- Iteratively combine the clusters containing the two closest items by adding an edge between them.
- Stop when there are k clusters.

This is exactly Kruskal's algorithm.

The "clusters" are the connected components that Kruskal's algorithm has created after a certain point.

Example of "single-linkage, agglomerative clustering."

Proof of Correctness

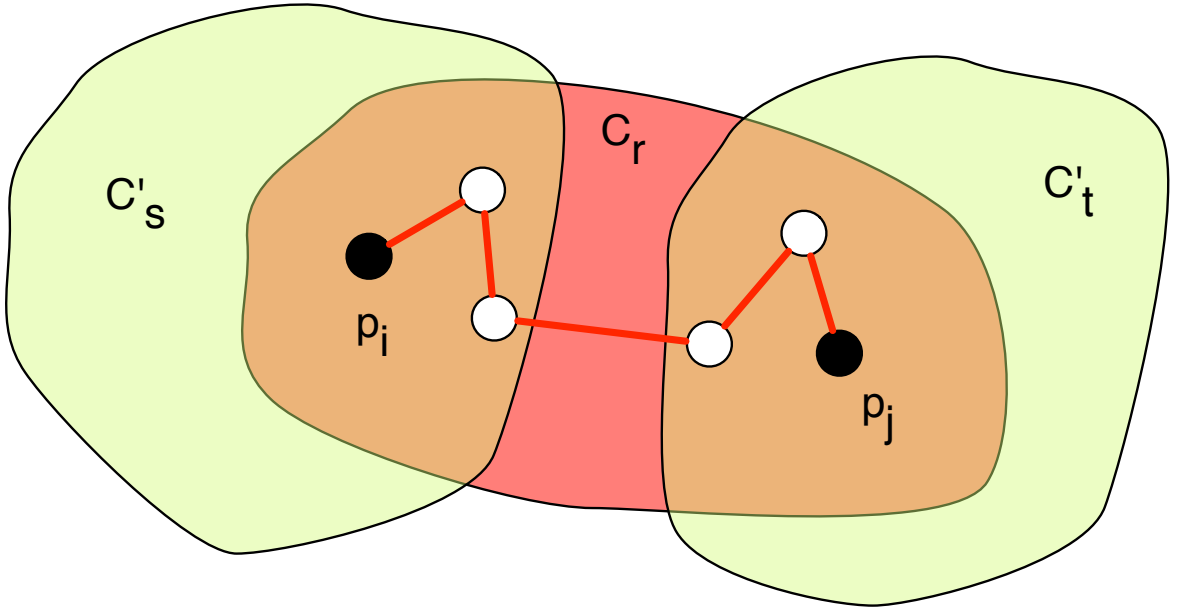
Another way too look at the algorithm: delete the $k-1$ most expensive edges from the MST.

The spacing d of the clustering C that this produces is the length of the $(k-1)^{\text{st}}$ most expensive edge.

Let C' be a different clustering. We'll show that C' must have the same or smaller separation than C.

Proof of correctness, 2

Since $C \neq C'$, there must be some pair p_i, p_j that are in the same cluster in C but different clusters in C'.



Together in $C \implies$ path P between p_i, p_j with all edges $\leq d$.

Some edge of P passes between two different clusters of C'.

Therefore, separation of $C' \leq d$.

Class So Far

6 lectures:

- Stable Marriage
- Topological Sort
- Detecting bipartite graphs
- Interval Scheduling
- Interval Partitioning
- Minimal Lateness Scheduling
- Optimal Caching
- Minimum Spanning Tree (3 Algs)
- Dijkstra's algorithm (proof of correctness)
- Matriods
- Min cost aboresences