Segmentation and greedy algorithms

Prof. Noah Snavely
CS1114
http://www.cs.cornell.edu/courses/cs1114

Cornell University
Computer Science

Administrivia

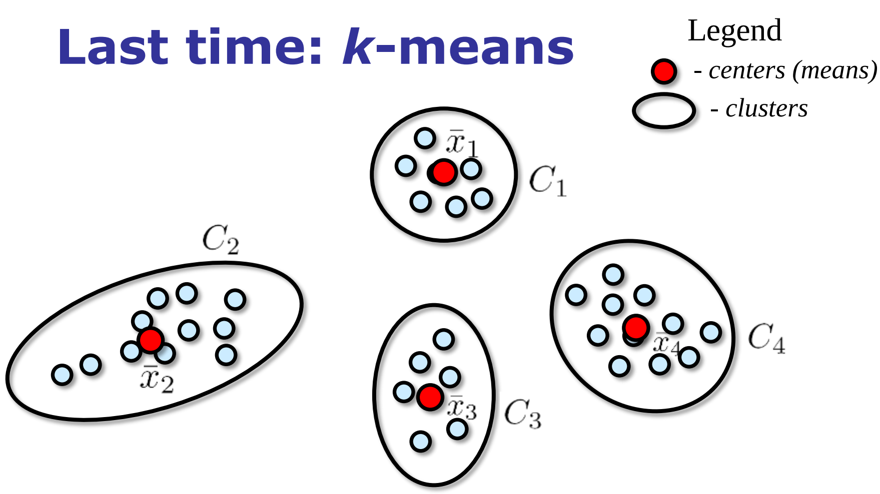
- A5P1 due tomorrow (demo slots available)
- A5P2 out this weekend, due 4/19

- Prelim 2 on Tuesday
  - Quizzes available Monday

- Midterm course evaluations

SIFT Matching Demo

Last time: *k*-means



k-means

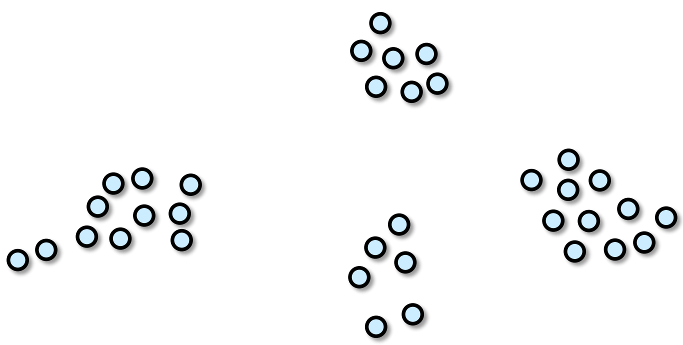
- Idea: find the centers that minimize the sum of squared distances to the points

- Objective:

Given input points $x_1, x_2, x_3, \ldots, x_n$, find the clusters $C_1, C_2, \ldots C_k$ and the cluster centers $\bar{x}_1, \bar{x}_2, \bar{x}_3, \ldots, \bar{x}_k$ that minimize

$$\sum_{j=1}^{k} \sum_{x_i \in C_j} |x_i - \bar{x}_j|^2$$

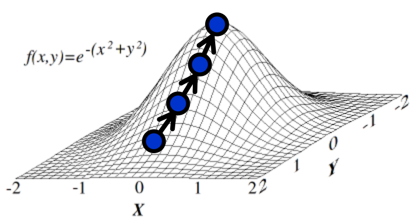
A greedy method for *k*-means

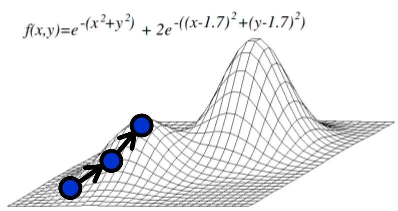


A greedy method for *k*-means

- Unfortunately, this doesn't work that well
- The answer we get could be **much** worse than the optimum
- However, if we change our objective (e.g., *k-centers*, then we get an answer within 2 times the cost of the best answer

"Hill climbing"





Back to k-means

- There's a simple iterative algorithm for *k*-means
  - Lloyd's algorithm

1. Start with an initial set of means
   - For instance, choose *k* points at random from the input set
2. Assign each point to the closest mean
3. Compute the means of each cluster
4. Repeat 2 and 3 until nothing changes

Lloyd's algorithm

Demo

Lloyd's algorithm

- Does it always terminate?
  - Yes, it will always *converge* to some solution
  - Might be a local minima of the objective function

$$\sum_{j=1}^{k} \sum_{x_i \in C_j} |x_i - \bar{x}_j|^2$$

  - Error decreases after every iteration
  - Error could be arbitrarily bad

Questions?

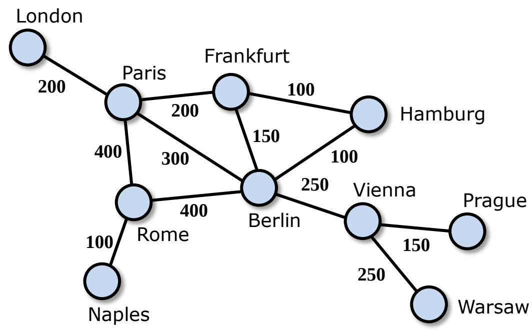
Possible algorithms

1. **Greedy** algorithms
   – Do what seems best at any given point
   – Example: making change
2. **Iterative** algorithms
   – Start with some answer, take a small step to improve it, repeat until it doesn't get better
   – Examples: Lloyd's algorithm for k-means, bubble sort, hill climbing

Where we are so far

- Greedy algorithms and iterative algorithms sometimes give the right answer (e.g., making change with U.S. currency)
- Some clustering objective functions are easier to optimize than others:
  – *k*-means → very hard
  – *k*-centers → very hard, but we can use a greedy algorithm to get within a factor of two of the best answer

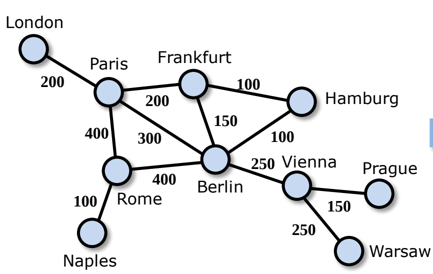
Back to graphs

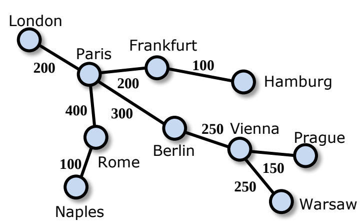


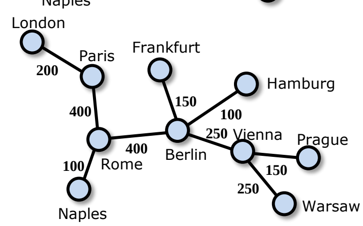
- We can also associate a *weight* with each edge (e.g., the distance between cities)

Spanning trees

- A spanning tree of a graph is a subgraph that (a) connects all the vertices and (b) is a tree



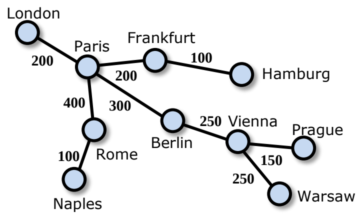


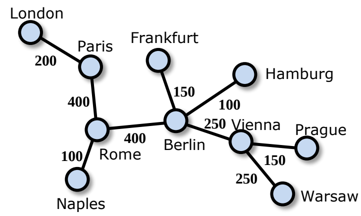


Q: How many edges are there in a spanning tree on* n *vertices?

Graph costs

- We'll say the *cost* of a graph is the sum of its edge weights



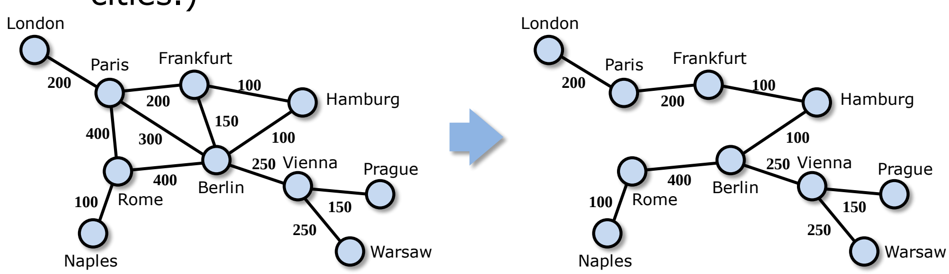


Cost = 200 + 200 + 100 +
400 + 300 + 100 +
250 + 150 + 250 = **1950**

Cost = 200 + 400 + 100 +
400 + 150 + 250 +
100 + 150 + 250 = **2000**

Minimum spanning trees

- We define the *minimum spanning tree* (MST) of a graph as the spanning tree with minimum cost
- (Suppose we want to build the minimum length of track possible while still connecting all the cities.)

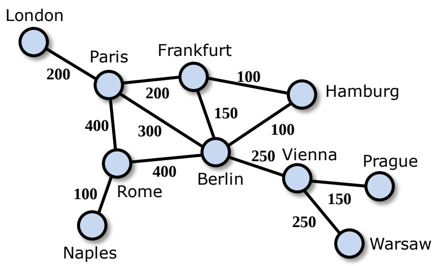


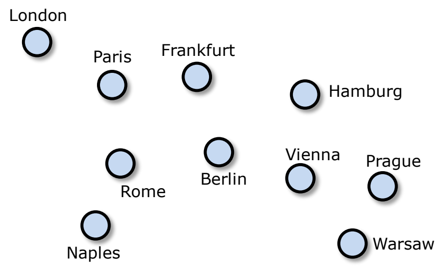
MST: Cost = 1750

(Eurorail needs to build 1750 mi of track at minimum)

Minimum spanning trees

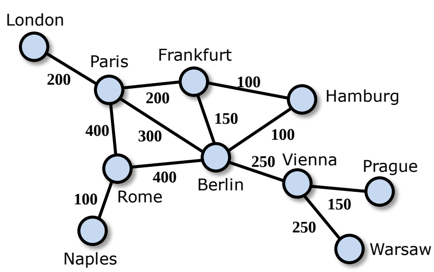
- This is an optimization problem where the objective function is the cost of the tree
- Can you think of a greedy algorithm to do this?

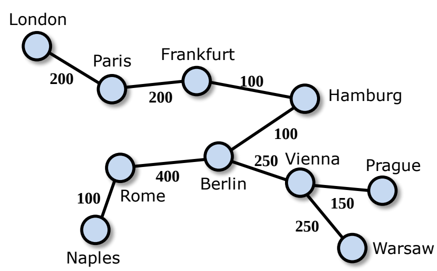




Minimum spanning tree

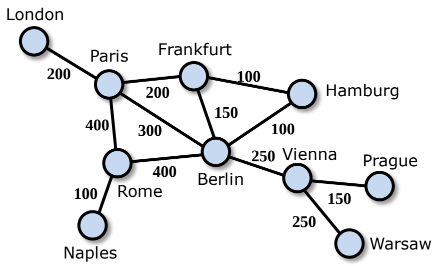
- Greedy algorithm:

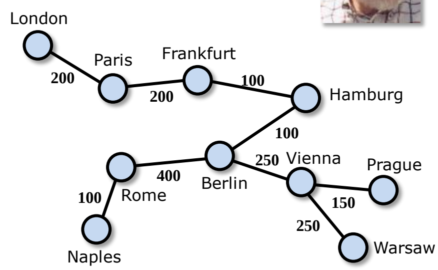




Minimum spanning tree

- This greedy algorithm is called **Kruskal's algorithm**





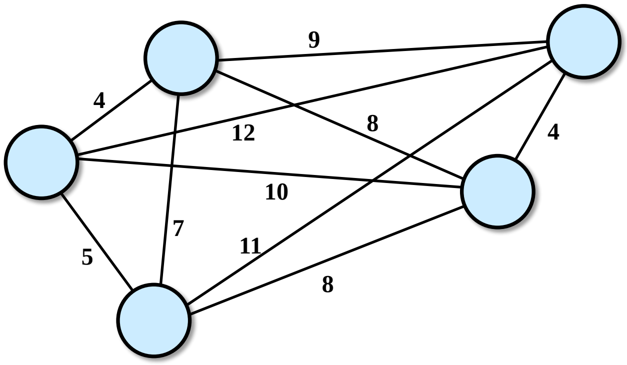
- Not that simple to prove that it gives the MST
- How many connected components are there after adding the k^{th} edge?

Kruskal's algorithm

- Start with an empty graph
- Sort edges by weight, in increasing order
- Go through each edge in order
  - If adding edge creates a cycle, skip it
  - Otherwise, add the edge to the graph

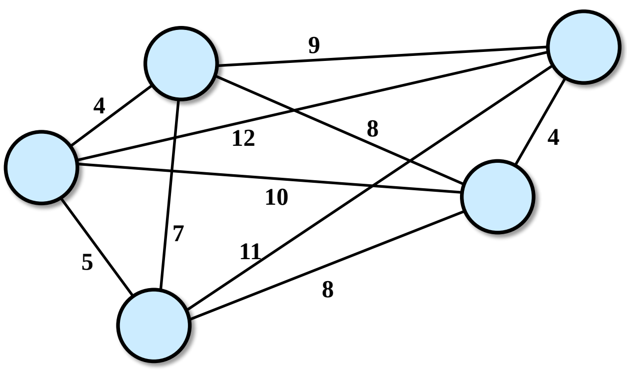
Back to clustering

- We can define the clustering problem on graphs



Clustering using graphs

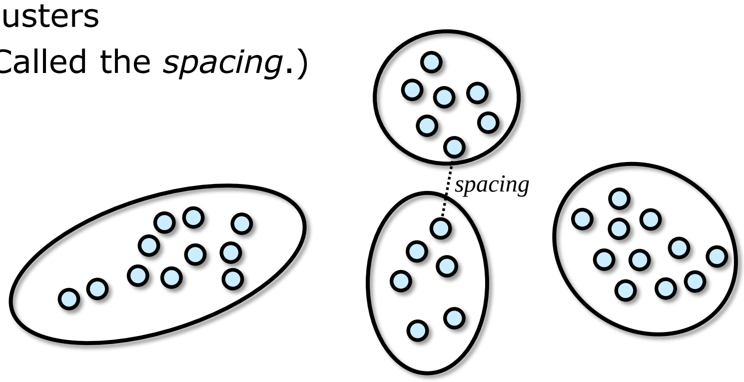
- Clustering → breaking apart the graph by cutting long edges



- Which edges do we break?

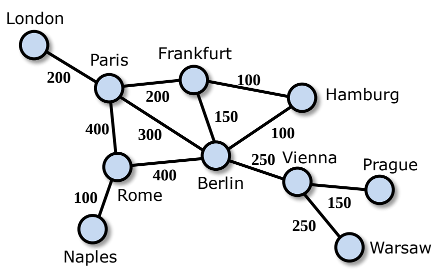
Spacing as a clustering metric

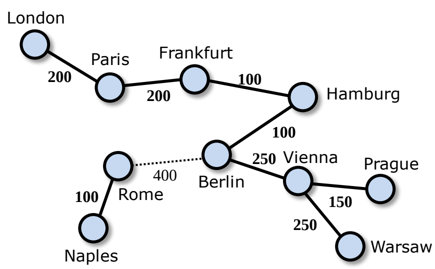
- Another objective function for clustering:
  - Maximize the *minimum* distance between clusters
  - (Called the *spacing*.)



Cool fact

- We compute the clusters with the maximum spacing during MST!
- To compute the best *k* clusters, just stop MST construction *k*-1 edges early



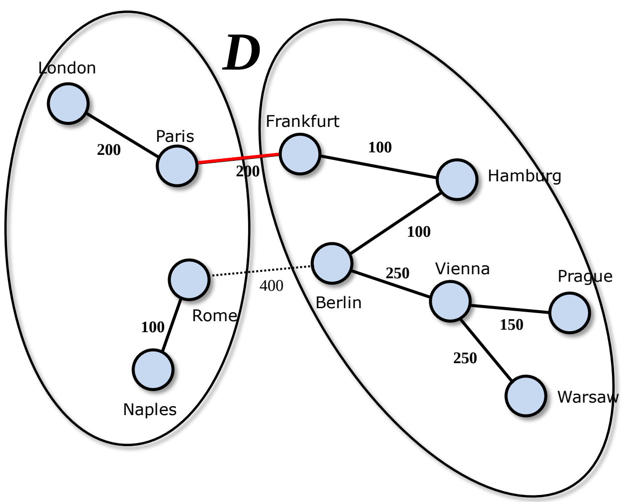


2 clusters with max spacing (=400)

Proof of cool fact

- Suppose this wasn't true – then someone could give us a different clustering with a bigger spacing
- Let *C* be our MST clustering, and let *D* be the purportedly better one
- There must be two nodes *u* and *v* in different clusters in *D* but in the same cluster in *C*
- There's a path between *u* and *v* in *C*, and at some point this path crosses a cluster boundary in *D*

Pictorial proof



Demo

- http://www.kovan.ceng.metu.edu.tr/~maya/kmeans/index.html

Where we are so far

- Greedy algorithms work sometimes (e.g., with MST)
- Some clustering objective functions are easier to optimize than others:
  - *k*-means → very hard
  - *k*-centers → very hard, but we can use a greedy algorithm to get within a factor of two of the best answer
  - maximum spacing → very easy! Just do MST and stop early (this gives exact answer)

Back to image segmentation

Questions?

Greedy algorithm for graph coloring?