Network Analysis

Dr. Shaina Race
Institute for Advanced Analytics

Spring 2018

Community Detection

i.e. Clustering

Clustering in Graphs

- Can still use classical algorithms (**k-means**) in most cases
  - **Edge weights must reflect similarity** and not distance
  - Use the **adjacency matrix like a data matrix**
  - The "observations" and "variables" are the same entities, but you simply characterize an observation by its similarity to others.
- Families of algorithms designed specifically for graphs
  - **Spectral** (Eigenvector) methods
  - **Modularity**
  - Minimum Spanning Trees
- Theorem: Nothing works best all the time!

Spectral Clustering

(Spectral ➔ Eigenvalues/Eigenvectors. Yay!)

Not just for Graphs!

- Keep in mind the following methods operate on a similarity matrix (i.e. adjacency matrix).

- If you develop a notion of similarity using traditional data, these methods can be useful for clustering any data!

The Laplacian Matrix

- Spectral methods typically use a Laplacian Matrix.

- Let **A** be an adjacency matrix for a graph (or a similarity matrix for some data)
- Let **D** be a diagonal matrix containing the degrees of each node:
  - **D** = diag{d_1,d_2, … ,d_n}
- The **Laplacian matrix** is defined as **L** = **D**-**A**

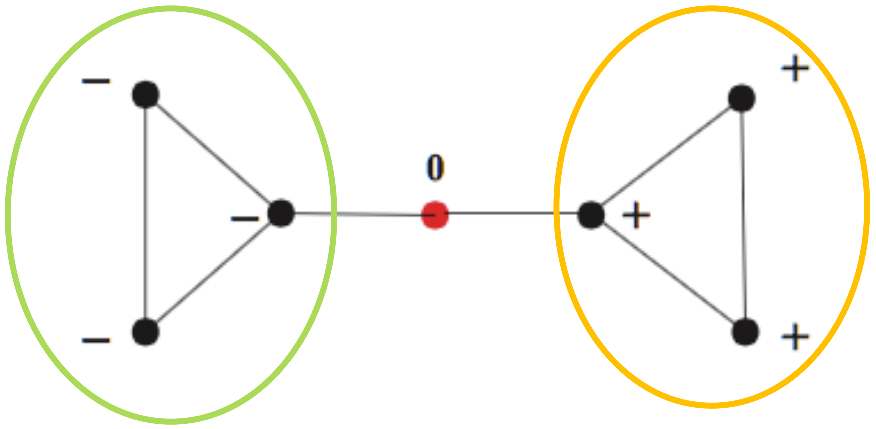
Simple Spectral Clustering

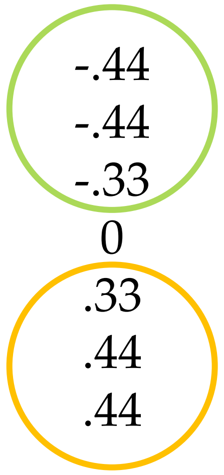
- The **Laplacian matrix** is defined as **L** = **D**-**A**
- The **Fiedler vector** is the eigenvector associated with the *second smallest* eigenvalue of **L**.

Baltimore Ravens' ex-Offensive Lineman John Urschel can tell you about it

Simple Spectral Clustering

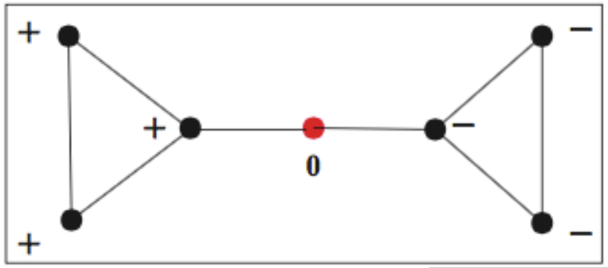
- Use the signs of the entries in the Fiedler vector.
  - Nodes associated with positive entries in one cluster
  - Nodes associated with negative entries in second cluster
  - Arbitrarily assign nodes associated with zero entries (often called **Articulation Points** – sometimes they are brokers)

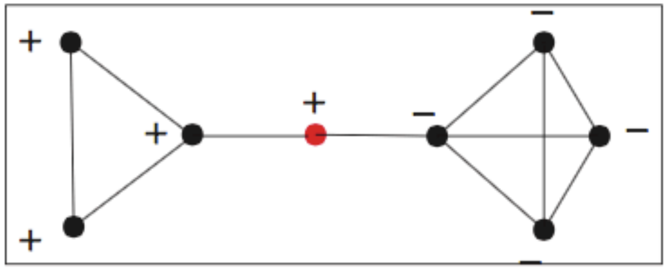


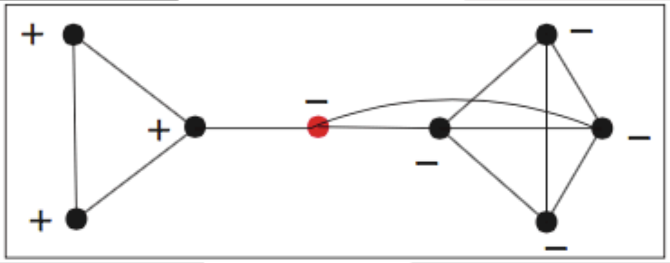


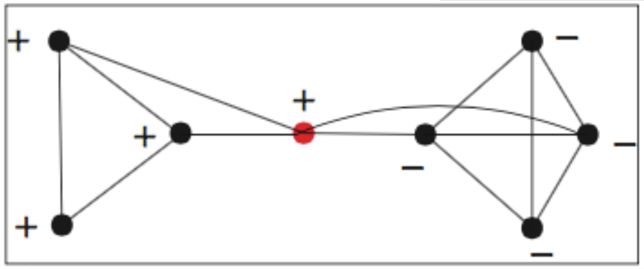
Exploring Articulation Points

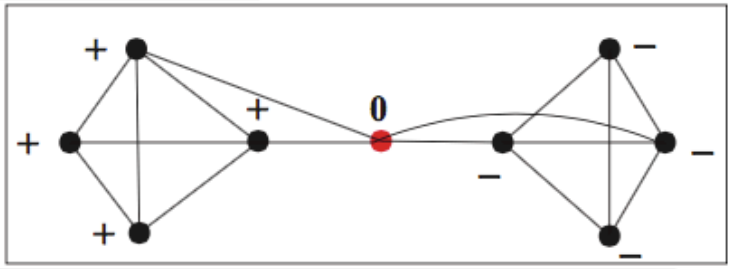
```
A=matrix(c(0,1,1,0,0,0,0,1,0,1,0,0,0,0,1,1,0,1,0,0,0,0,0,1,0,
1,0,0,0,0,0,1,0,1,1,0,0,0,0,1,0,1,0,0,0,0,1,1,0),nrow=7)
library("igraph")
g=graph_from_adjacency_matrix(A)
plot(g)
D=diag(rowSums(A))
L-D-A
spectrum=eigen(L)
spectrum$vectors[,6]
```

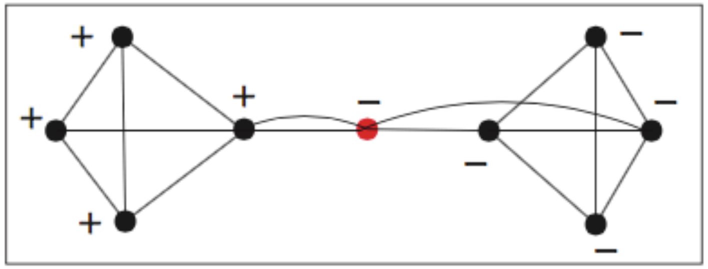

+
+
+
0
−
−
−

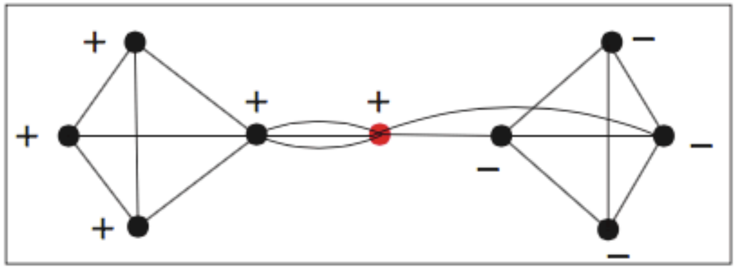

+
+
+
+
−
−
−
−


+
+
+
−
−
−
−
−


+
+
+
+
−
−
−
−

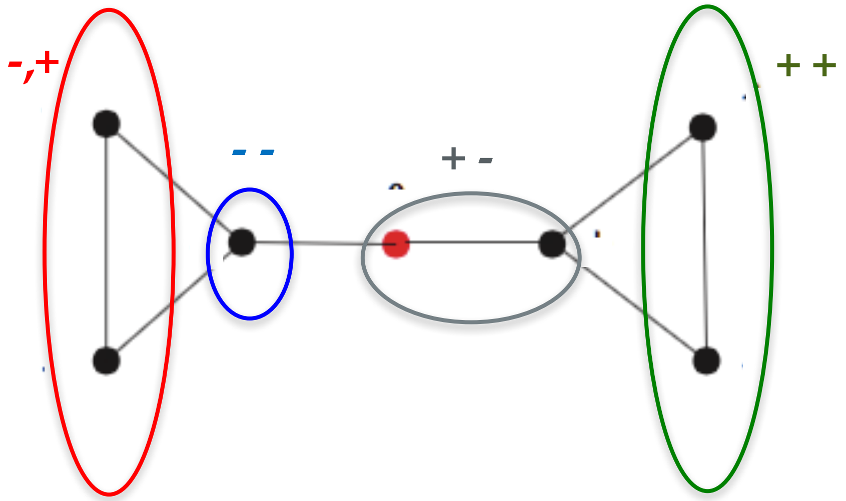

+
+
+
+
0
−
−
−
−


+
+
+
+
−
−
−
−
−


+
+
+
+
+
−
−
−
−

Simple Spectral Clustering

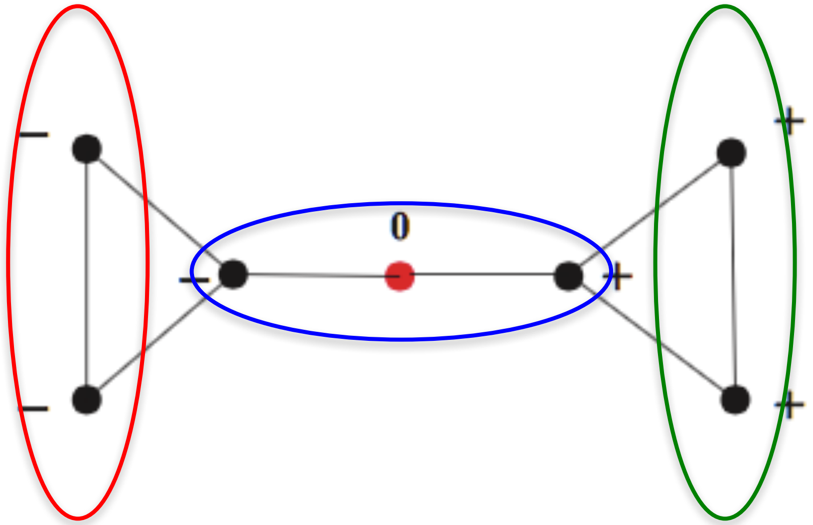
- How to get more than two clusters? (Two Ways)
  - Repeat process on each cluster.
  - Use additional eigenvectors (Two Ways)
    - Use the sign patterns (shown below)
    - Cluster the rows of the eigenvectors with k-means (Next Slide)



$\mathbf{v}_6$	$\mathbf{v}_5$
-.44	0.28
-.44	0.28
-.33	-0.16
0	-0.79
.33	-0.16
.44	0.28
.44	0.28

Simple Spectral Clustering

- Use k-means to cluster the rows of the eigenvectors
    - (Get to choose k in this case)



$\mathbf{v}_6$	$\mathbf{v}_5$
-.44	0.28
-.44	0.28
-.33	-0.16
0	-0.79
.33	-0.16
.44	0.28
.44	0.28

Advanced Spectral Clustering

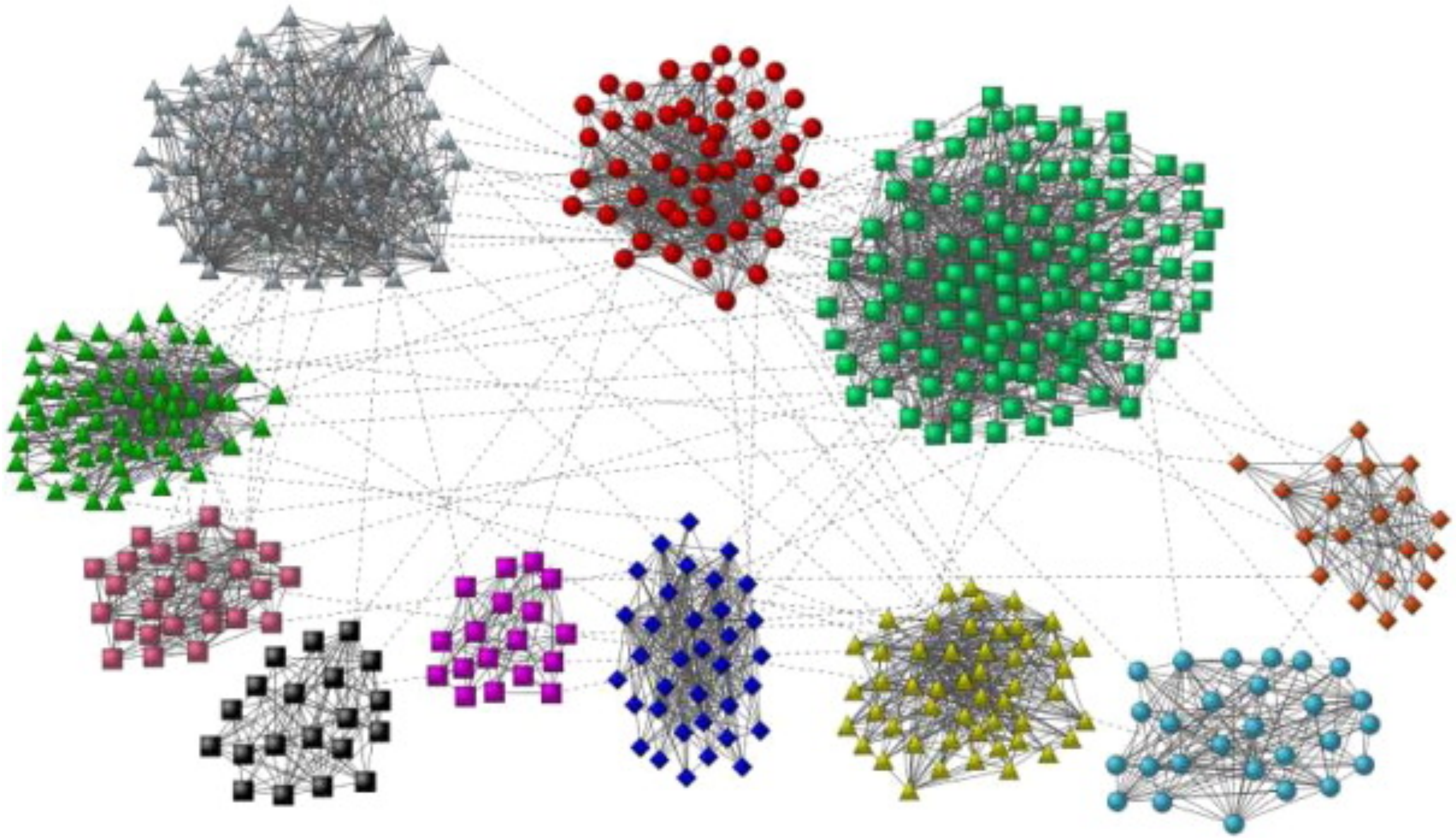
- NCUT
- Ratio Cut
- Normalized Spectral Clustering
- …Long list of algorithms

- All involve the Laplacian Matrix, typically normalized in different ways, and k-means run on Eigenvectors

Modularity Maximization

- Currently the **most popular** algorithm for community detection.
- Developed in 2006 by Mark Newman (UMichigan)
- Algorithm **Intuition**:
  - Compare the observed network to what you would expect to find at random.
  - Where are there more edges than expected?
  - These areas may define communities.

Modularity Maximization

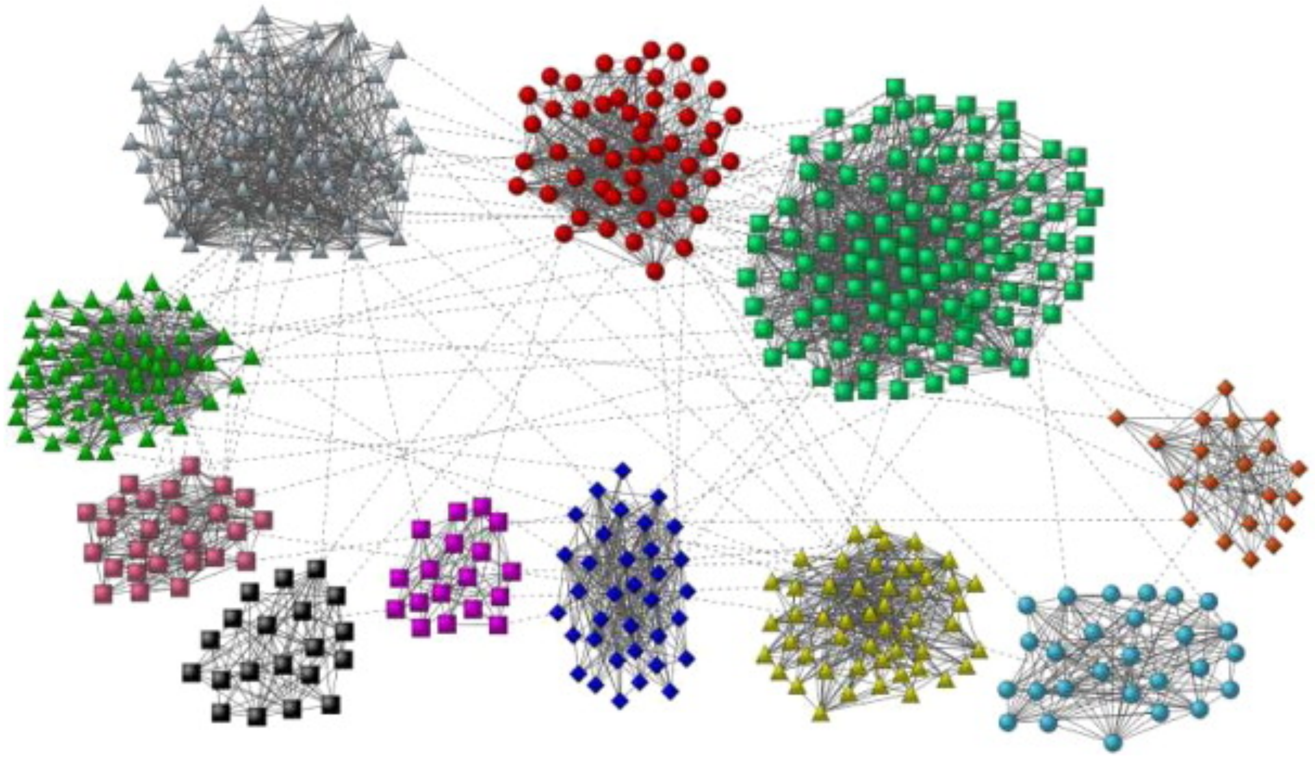


Modularity

- **Modularity** is a number that describes the ***extent to which given groups form communities in a graph***.
- Fraction of edges within groups minus the *expected* fraction if edges were distributed at random.
- Number in range [-1, 1)
  - **negative => random partition**
    - (We'd expect to find *more* edges within our groups if they were distributed at random)
  - **nearer 1 => better communities (components)**
    - (We see far more edges within our groups than we'd expect to find at random)

Modularity Maximization

Picks the partitioning of the vertices that maximizes the modularity.

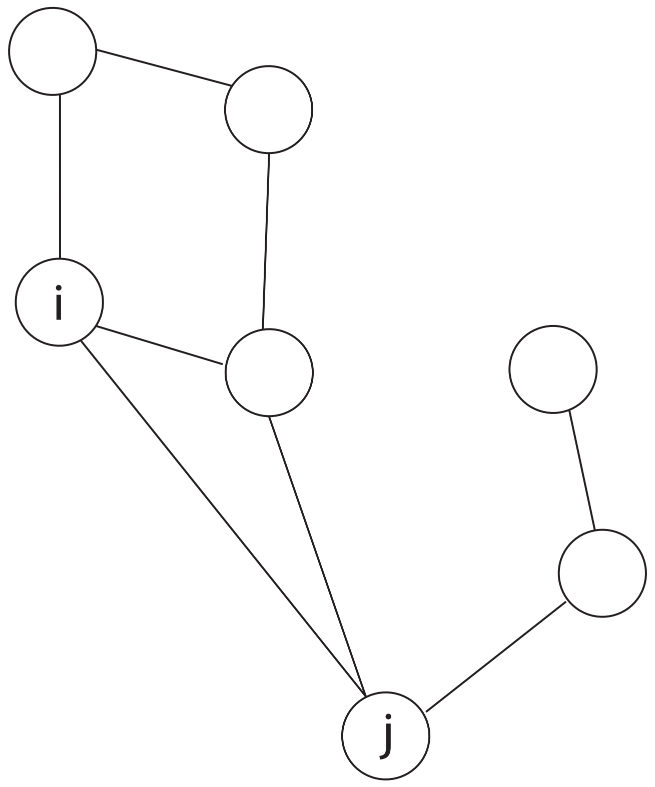


Modularity Maximization

- Let **A** be the adjacency matrix of the graph
- Let **P** be a matrix containing the expected number of edges between each vertex.

$$P_{ij} = \frac{d_i d_j}{\sum_k d_k}$$

Modularity Maximization



P(stub is connected to i) = 3/16
P(stub is connected to j) = 3/16

If i and j are independent,
P(i <-> j) = P(i) • P(j)

So the expected value of the number of complete edges between i and j is
P(i)•P(j) • 16.

Modularity Maximization

$$P_{ij} = \frac{d_i d_j}{\sum_k d_k}$$

- **B** = **A-P** is the modularity matrix
- The first eigenvector of the modularity matrix partitions the graph in two components.
- Repeat procedure on each component until the first eigenvalue is negative.

Algorithm 13 Modularity Procedure for Network Community Detection (Newman) [94]

```
**Input:** $n \times n$ adjacency matrix $\mathbf{A}$ for an undirected graph to be partitioned
1. Let $d_i$ be the $i^{th}$ row sum of $\mathbf{A}$. Let $d = \sum_{i=1}^{n} d_i$
2. Form the matrix $\mathbf{P}$ with $\mathbf{P}_{ij} = d_i d_j / d$.
3. Form the modularity matrix $\mathbf{B} = \mathbf{A} - \mathbf{P}$.
4. Compute the largest eigenvalue $\lambda_1$ and corresponding eigenvector $\mathbf{u}_1$ of $\mathbf{B}$.
5. If $\lambda_1 < 0$, stop. There is no partition of this graph.
6. Otherwise partition the vertices of the graph into 2 clusters as follows
       $C_1 = \{i : \mathbf{u}_1(i) < 0\}$
       $C_2 = \{i : \mathbf{u}_1(i) \geq 0\}$        (3.11)
7. Determine further partitions by extracting the rows and columns of the original adjacency matrix corresponding to the vertices in each cluster to form $\mathbf{A}'$ and repeat the algorithm with $\mathbf{A}'$ until each created cluster fails to partition in step 5.
**Output:** Final clusters.
```

Modularity Maximization

- Advantages
  - **Automatically determines number of clusters**
  - **Intuitive** rationale for/definition of a community
  - **Easy** to program and compute
- Disadvantages
  - Node can belong to only one community **(hard clustering)**
  - If first eigenvalue of modularity matrix is negative – no clusters.
    - (could be advantage)

Minimum Spanning Trees

An alternative

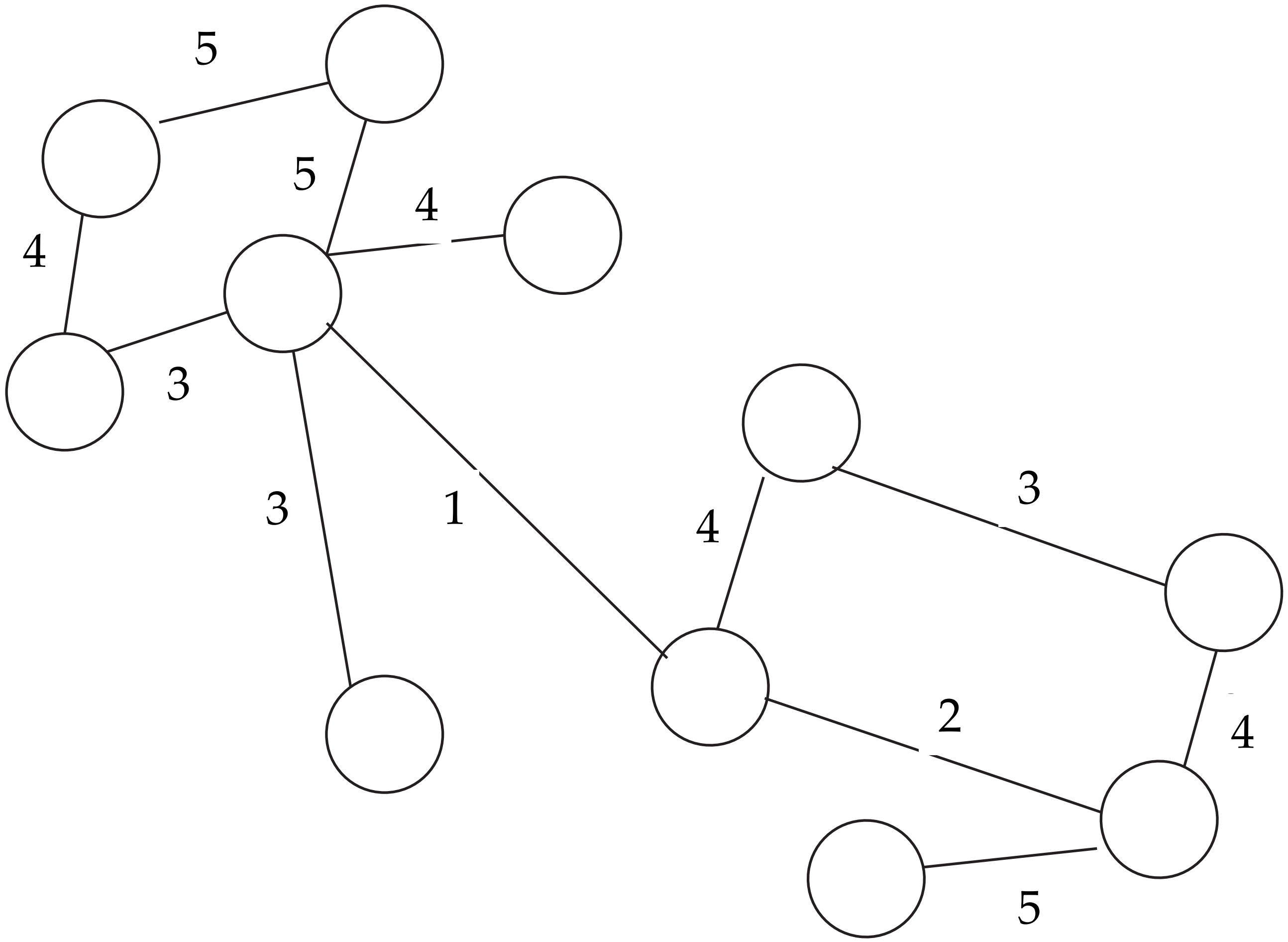
Minimum Spanning Trees

(or maximal spanning trees in the case of network similarity)

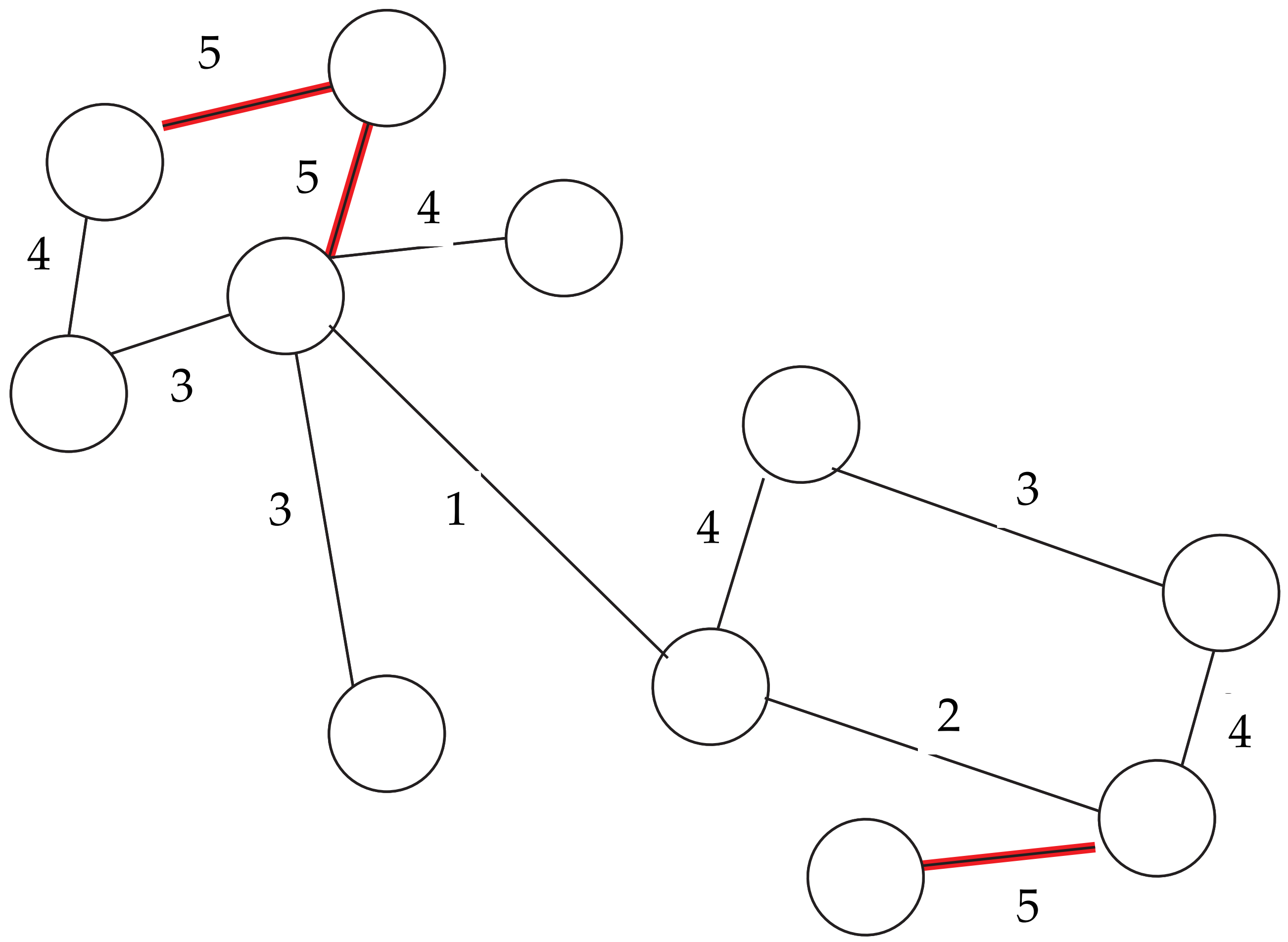
- Equivalent to **Single Linkage hierarchical clustering**.
- Creates a **tree** (graph with no cycles) that connects every node.
- Cutting all edges of the tree whose weight doesn't meet a pre-specified threshold will result in clusters.
- Changing threshold changes the number of clusters.

Building the *Maximum* Spanning Tree

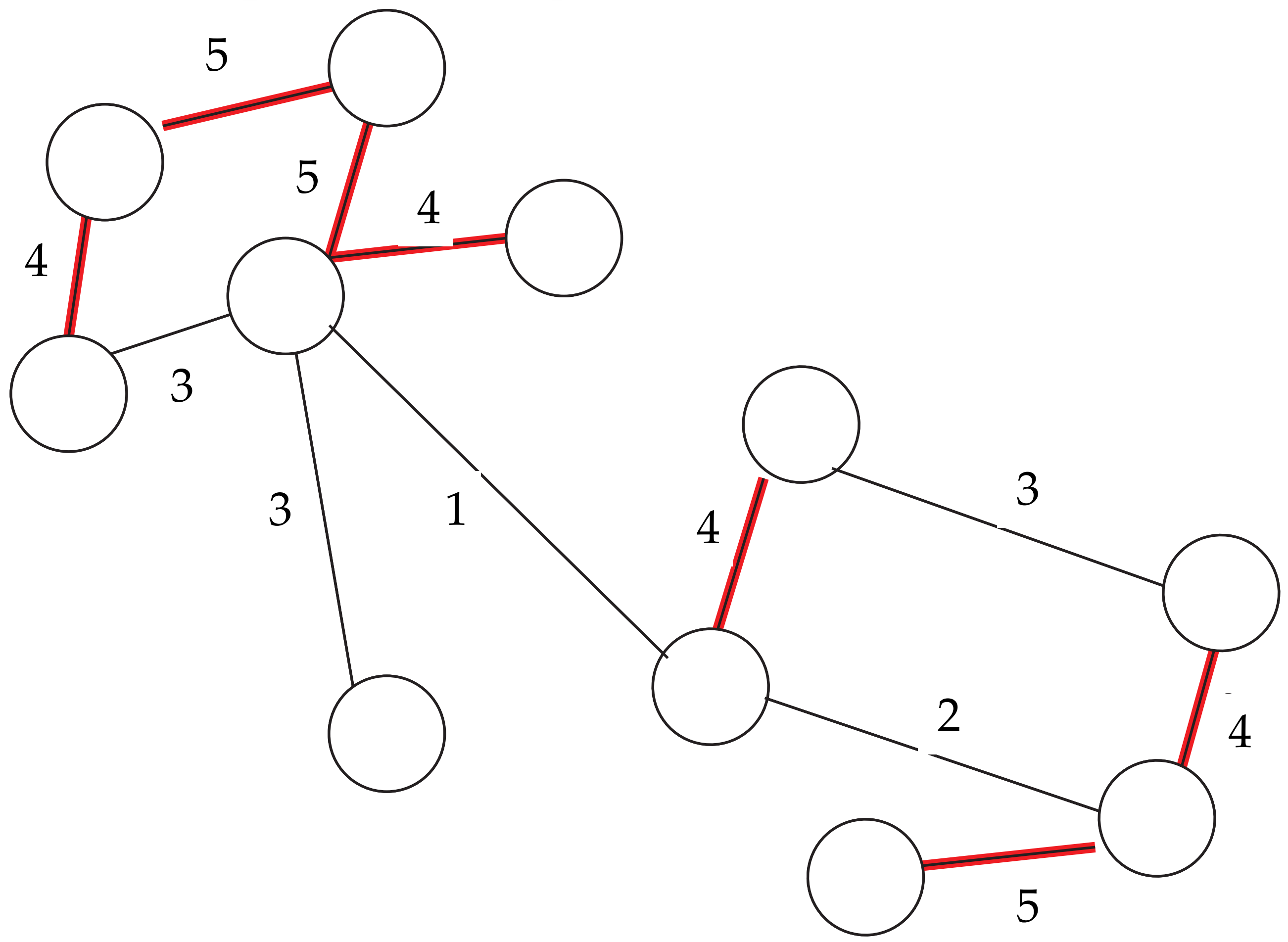
(When edge weights reflect similarity)



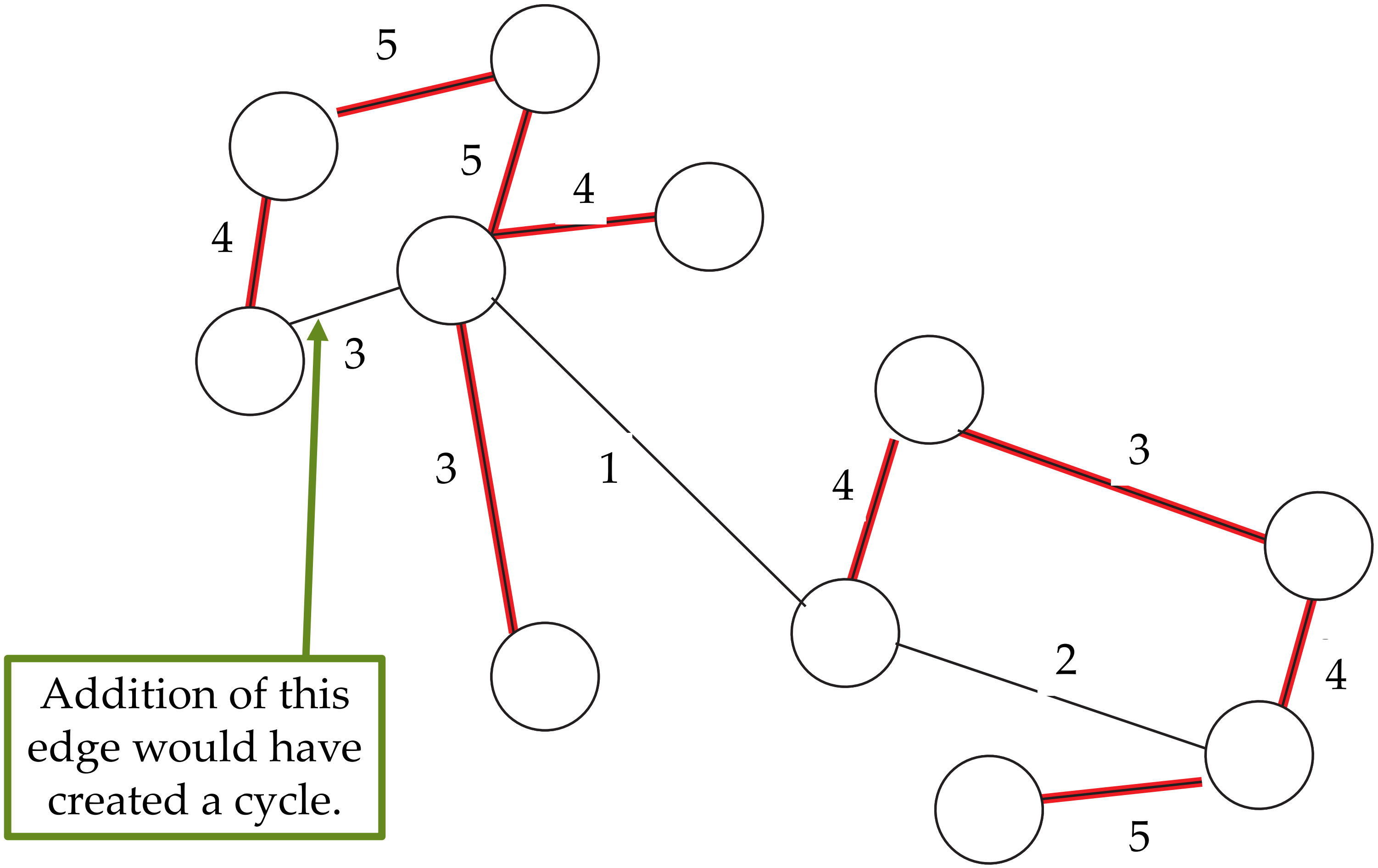
Add the most similar edges to the spanning tree, as long as no cycles are created. Repeat.



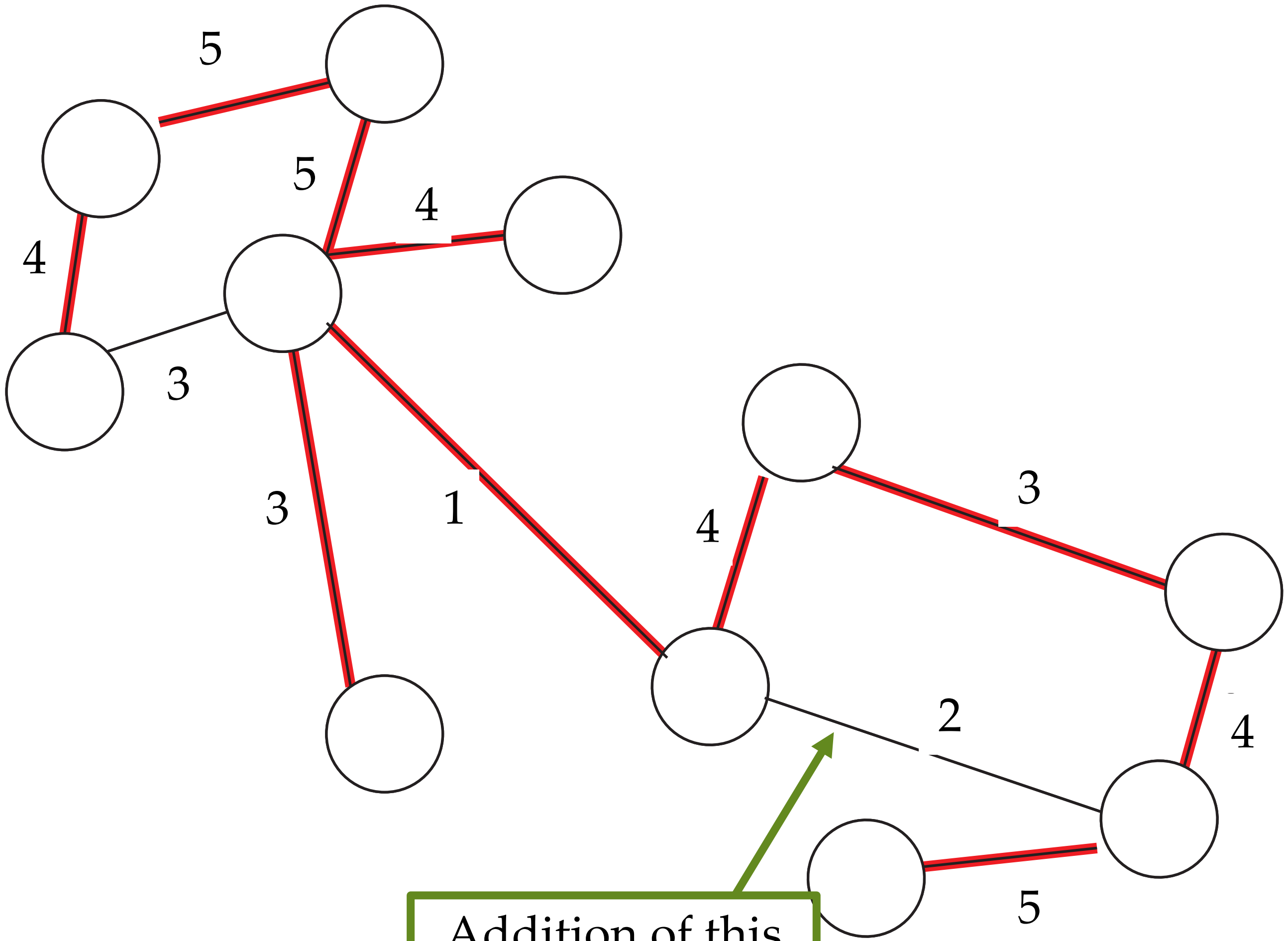
Add the most similar edges to the spanning tree, as long as no cycles are created. Repeat.



Add the most similar edges to the spanning tree, as long as no cycles are created. Repeat.



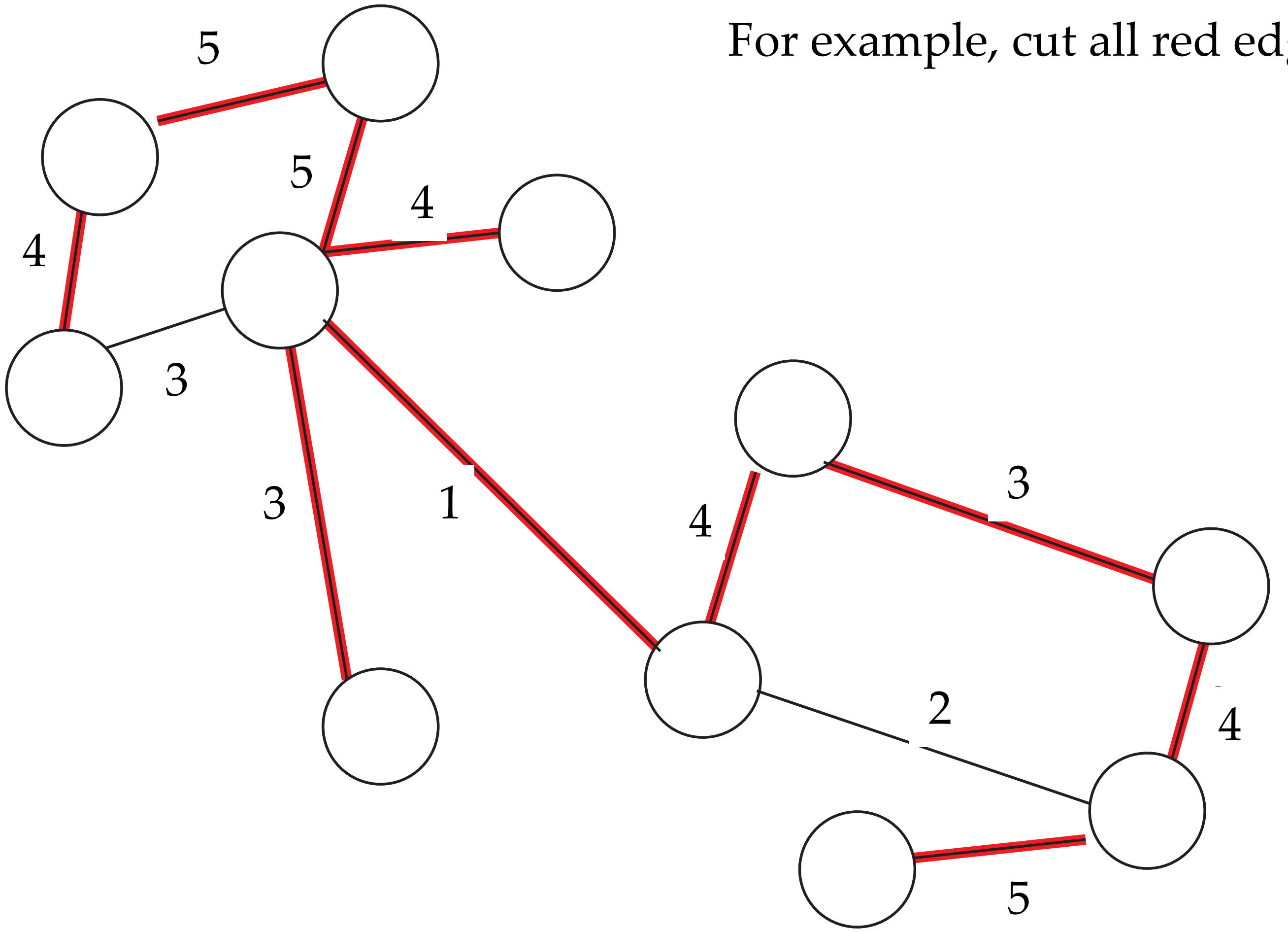
Add the most similar edges to the spanning tree, as long as no cycles are created. Repeat.



Addition of this edge would have created a cycle.

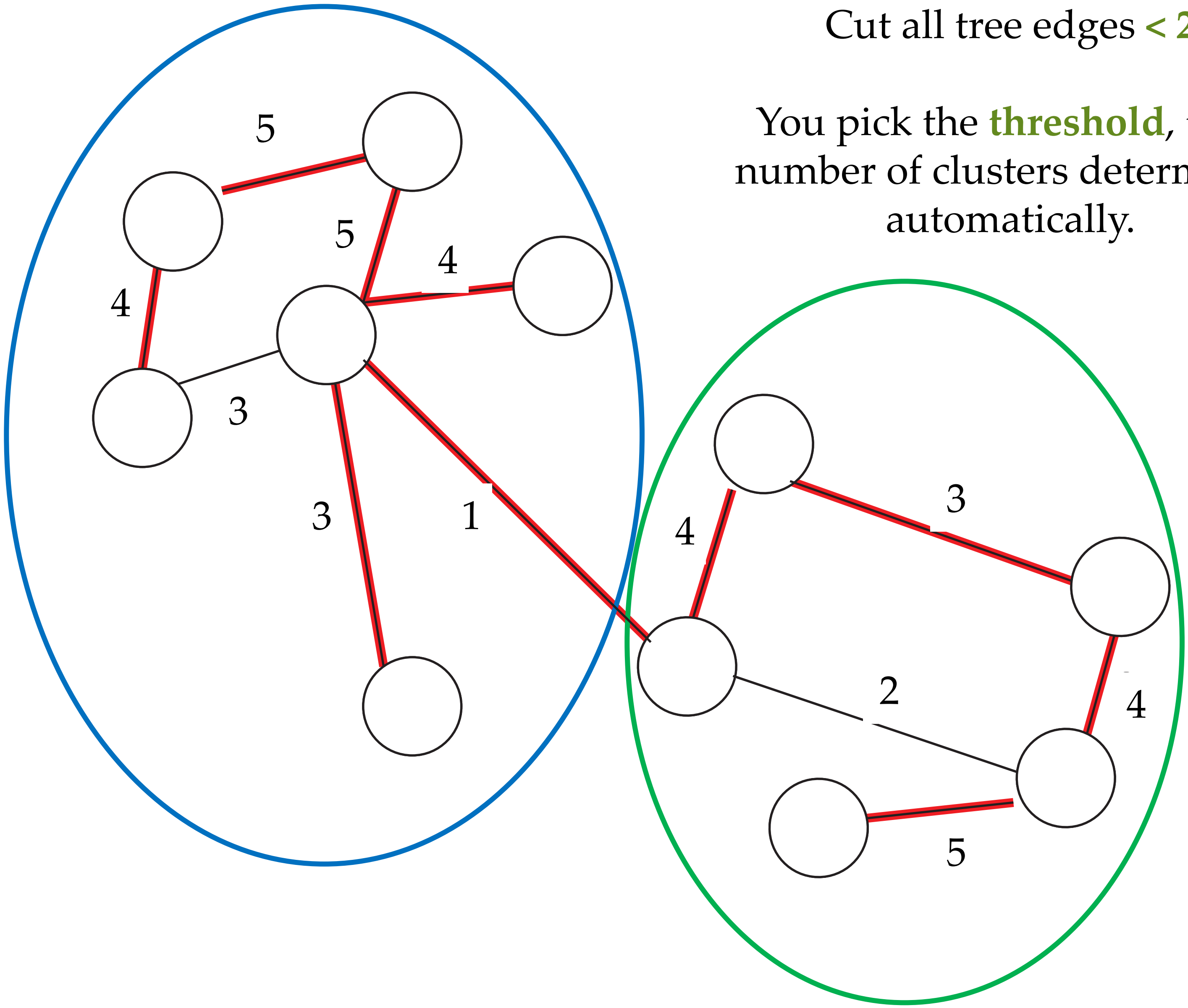
Cut the tree at some threshold.

For example, cut all red edges < 2



Cut all tree edges **< 2**

You pick the **threshold**, then number of clusters determined automatically.



Ensemble Clustering

- Try many different clustering algorithms
- (Or even k-means with different starting points)
- Create a network where the weight of the edge is the number of times that obs *i* was clustered with obs *j*.
- Partition the resulting network.