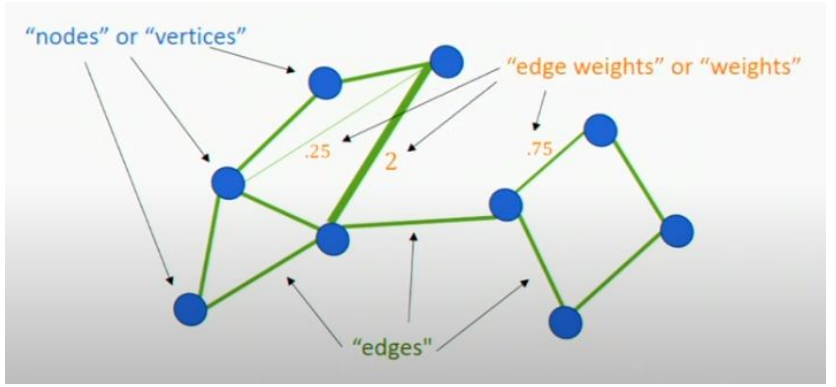
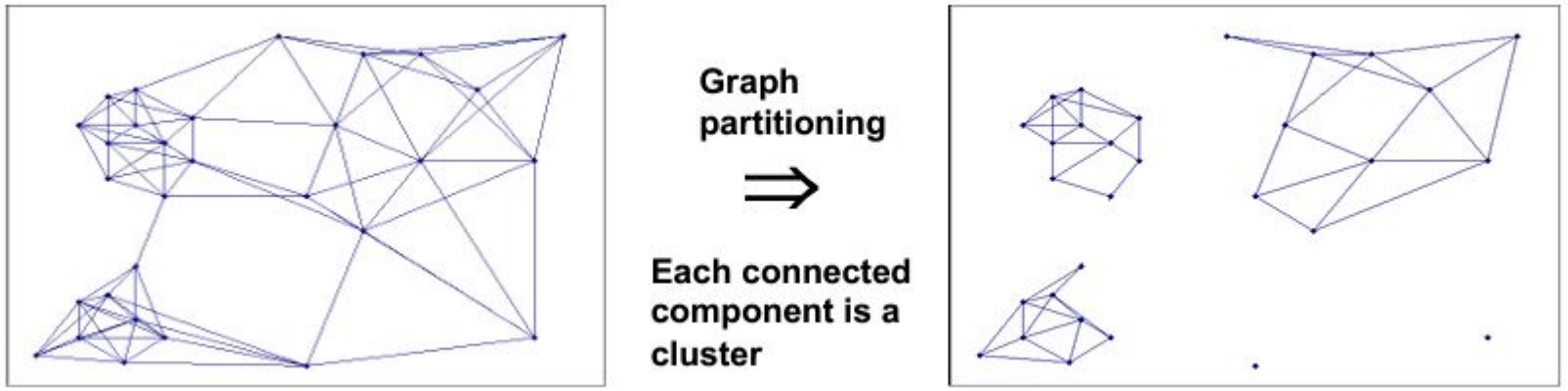
Similarity Graphs and Cut-based Clustering

Graph based clustering

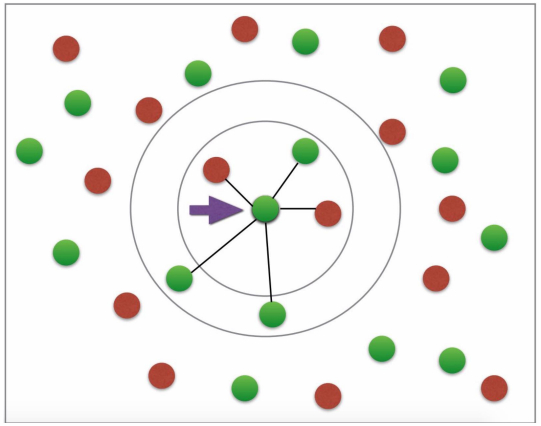


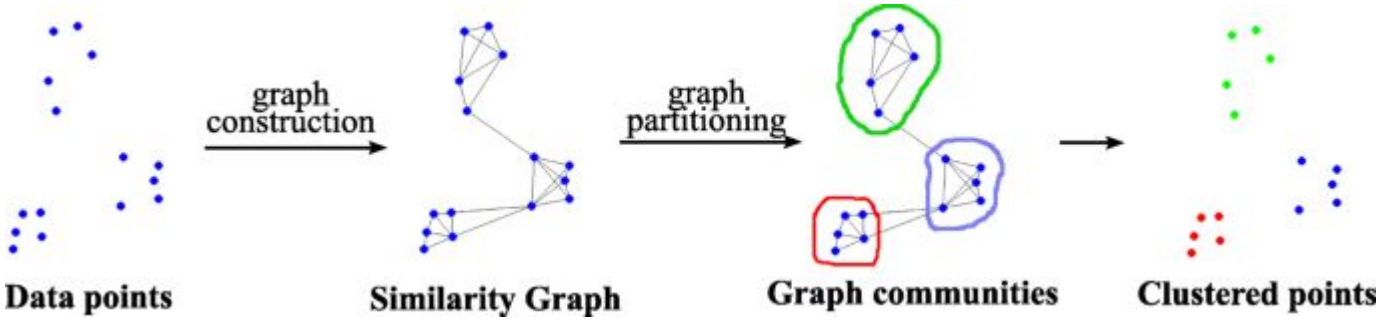
- Graph clustering is a branch of unsupervised learning, partitioning nodes in a graph into cohesive groups (clusters) based on their common characteristics
- Transform the data into a graph representation
  - where each element to be clustered is represented as a node and the distance between two elements is modeled by a certain weight on the edge linking the nodes.
- **Vertices** are the data points to be clustered
- **Edges** are weighted based on similarity between data points

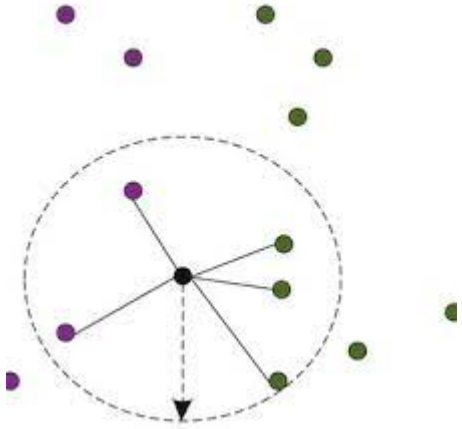


Graph Construction

- K-NN Graph
- Epsilon Neighborhood Graph



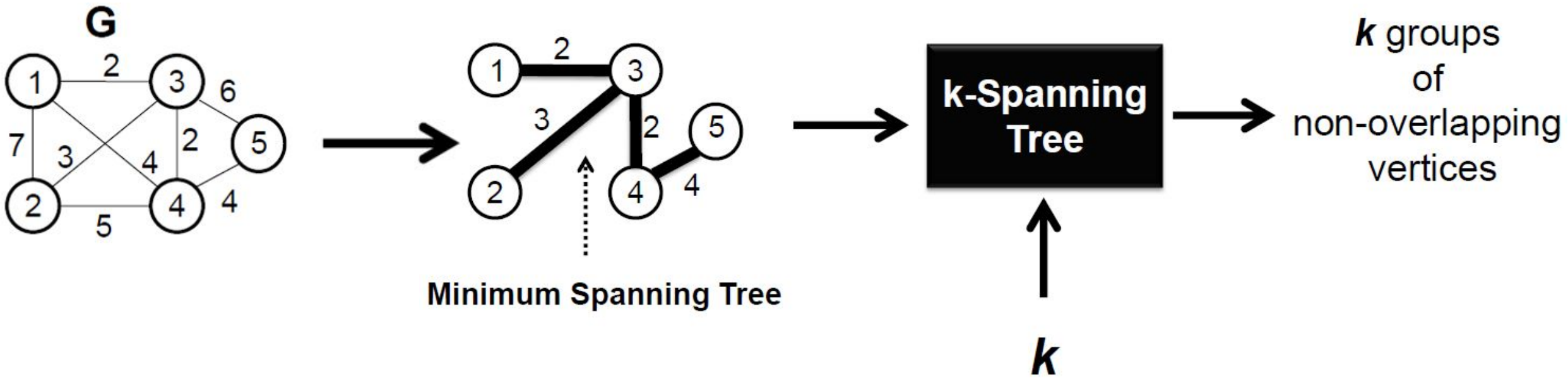




Applications

- **Social network analysis:** These algorithms help identify communities, influential individuals, and opinion leaders in social networks.
- **Recommendation Systems**: The algorithm helps to group similar objects, people, or items based on user behavior. Recommendation systems use this data to help users discover new accounts and receive relevant ads.
- **Biological network analysis**: Protein-protein interaction networks and gene expression networks are computationally represented as graphs. This lets researchers gain insights into their components' roles and interactions.
- **Security and fraud detection**: Graph clustering algorithms can detect outliers or anomalies within networks.

Graph based clustering - MST

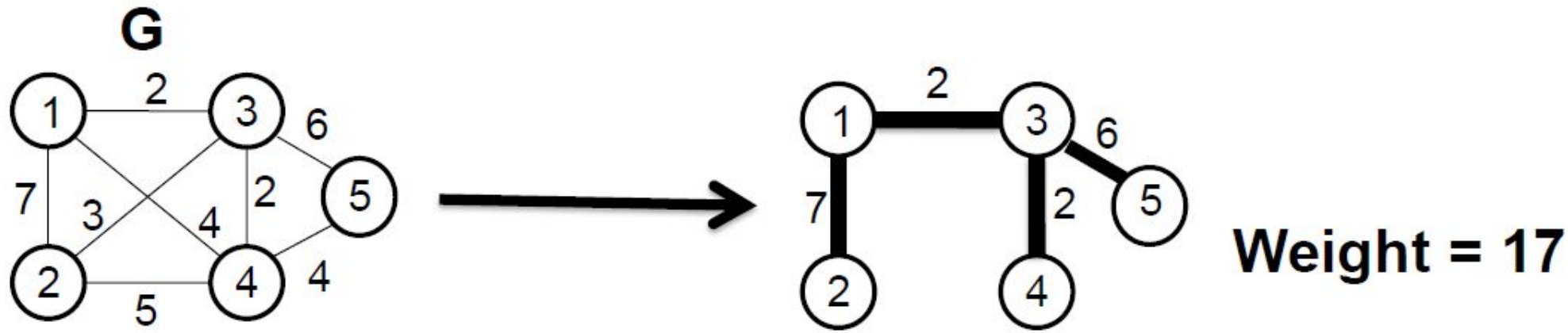


STEPS:

- **Obtains the Minimum Spanning Tree (MST) of input graph G**
- **Removes k-1 heaviest edges from the MST**
- **Results in k clusters**

Spanning Tree

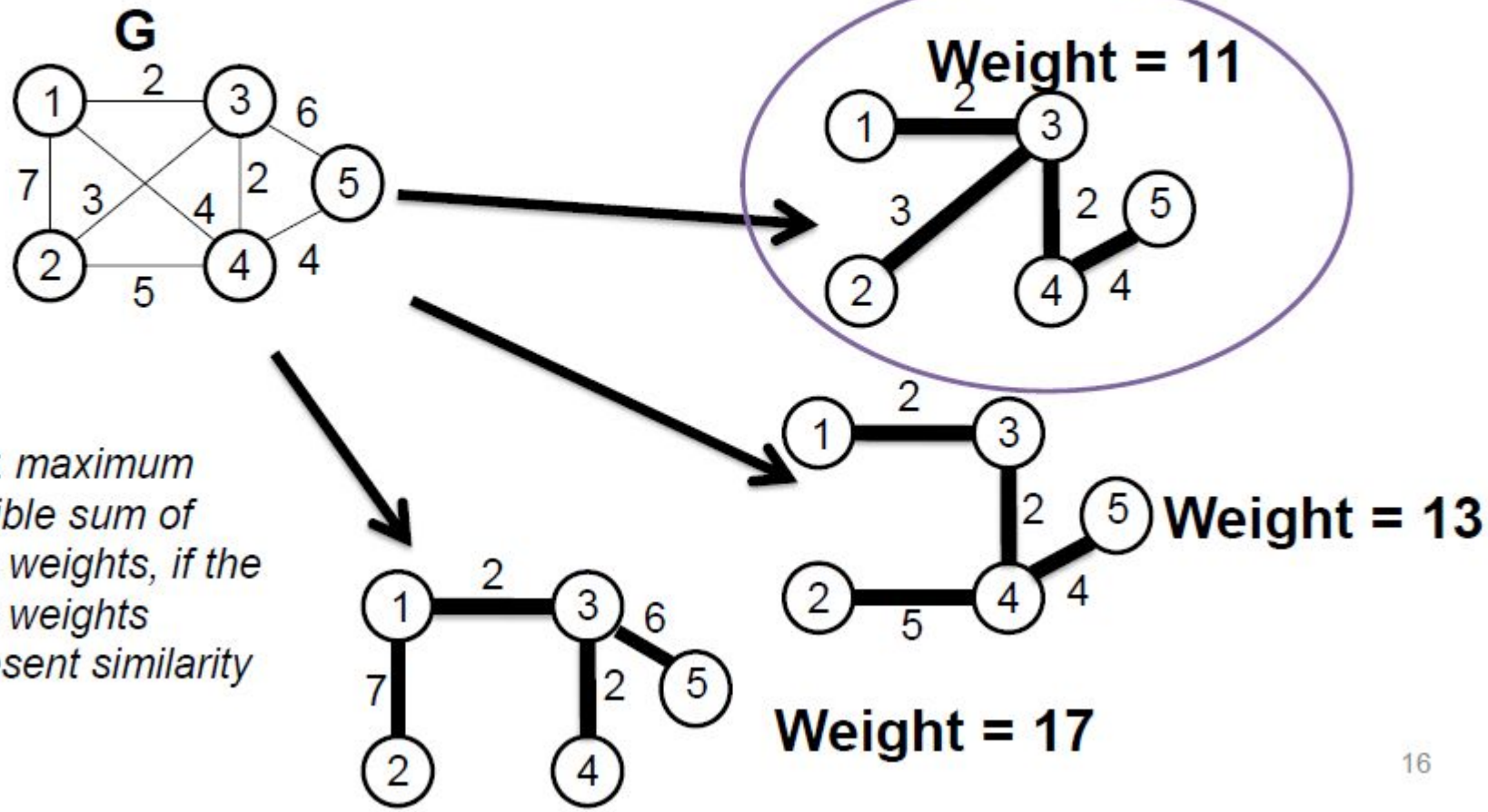
A connected subgraph with no cycles that includes all vertices in the graph



Note: *Weight can represent either distance or similarity between two vertices or similarity of the two vertices*

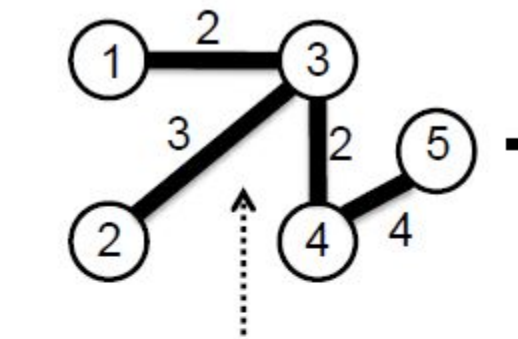
Minimum spanning tree

The spanning tree of a graph with the minimum possible sum of edge weights, if the edge weights represent distance



Note: *maximum possible sum of edge weights, if the edge weights represent similarity*

k-spanning tree

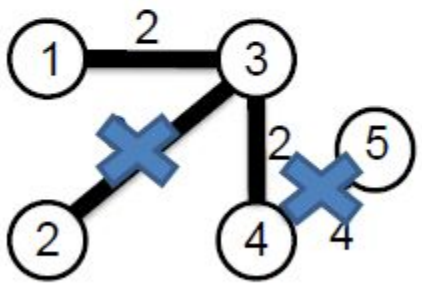


Minimum Spanning Tree

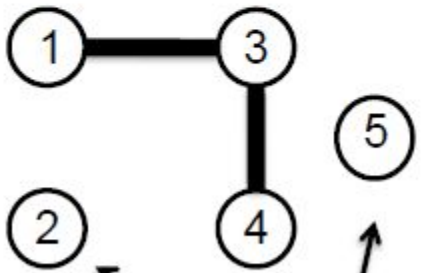
Remove k-1 edges with highest weight

Note: *k – is the number of clusters*

E.g., k=3



E.g., k=3



3 Clusters

Cut-based Graph Clustering

- Two things needed:

1.An objective function to determine what would be the best way to “cut” the edges of a graph

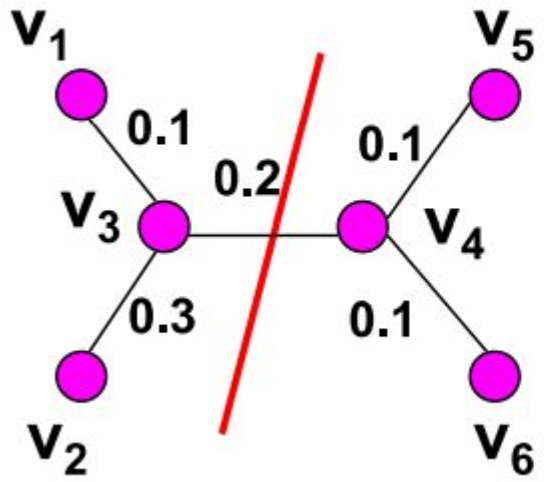
2.An algorithm to find the optimal partition (optimal according to the objective function)

Objective function for cut-based partitioning

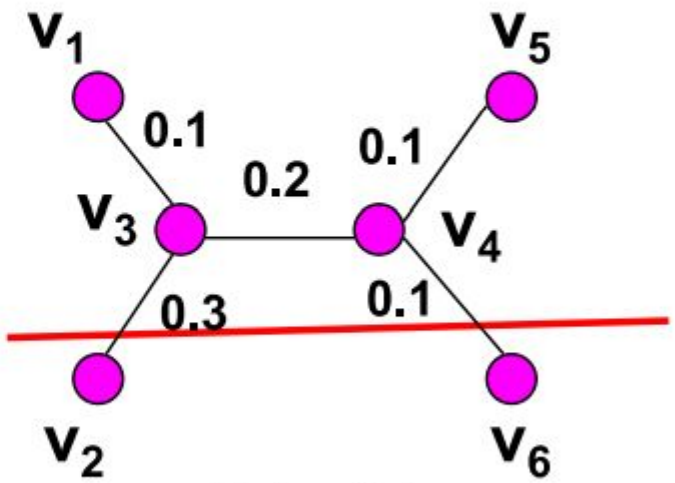
- Suppose we want to partition the set of vertices V into two sets: V_1 and V_2
- One possible objective function is to minimize graph cut

$$\mathrm{Cut}(V_1, V_2) = \sum_{\substack{i \in V_1, \\ j \in V_2}} w_{ij}$$

w_{ij} is weight of the edge between nodes i and j



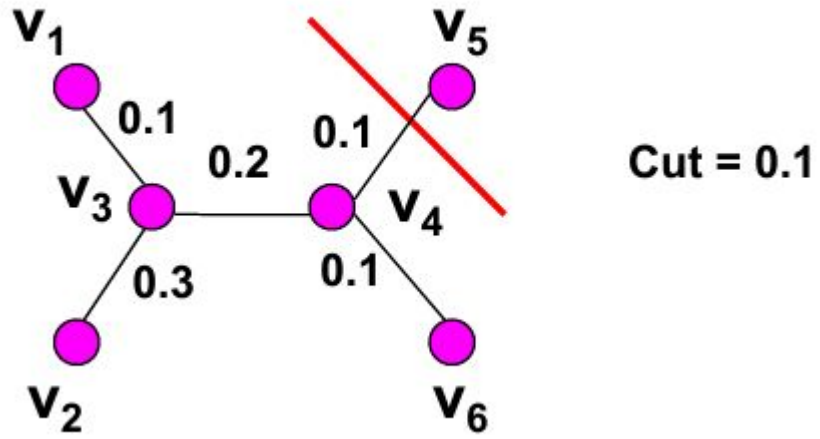
Cut = 0.2



Cut = 0.4

Limitation of minimizing graph cut

- The optimal solution might be to split up a single node from the rest of the graph! Not a desirable solution



Balanced clusters

- We should not only minimize the graph cut; but also look for “balanced” clusters

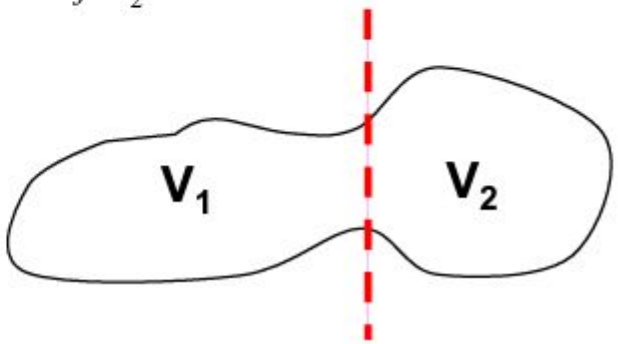
$$\text{Ratio cut}(V_1, V_2) = \frac{\text{Cut}(V_1, V_2)}{|V_1|} + \frac{\text{Cut}(V_1, V_2)}{|V_2|}$$

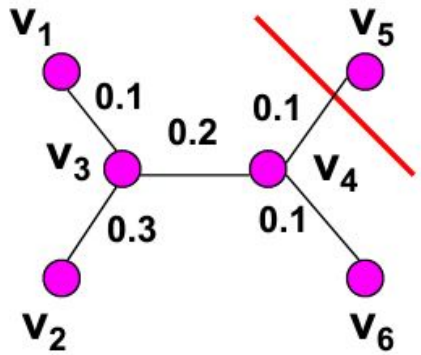
$$\text{Normalized cut}(V_1, V_2) = \frac{\text{Cut}(V_1, V_2)}{\sum_{i \in V_1} d_i} + \frac{\text{Cut}(V_1, V_2)}{\sum_{j \in V_2} d_j}$$

$$\text{where } d_i = \sum_j w_{ij}$$

V_1 and V_2 are the set of nodes in partitions 1 and 2

$|V_i|$ is the number of nodes in partition V_i

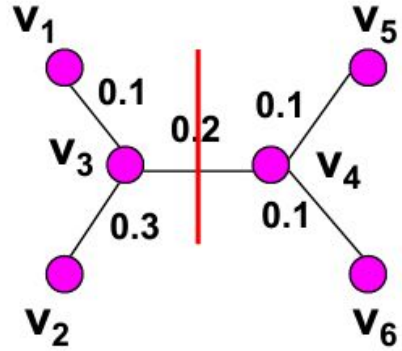




Cut = 0.1

Ratio cut = 0.1/1 + 0.1/5 = 0.12

Normalized cut = 0.1/0.1 + 0.1/1.5
= 1.07

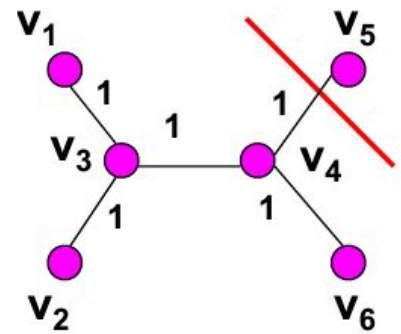


Cut = 0.2

Ratio cut = 0.2/3 + 0.2/3 = 0.13

Normalized cut = 0.2/1 + 0.2/0.6
= 0.53

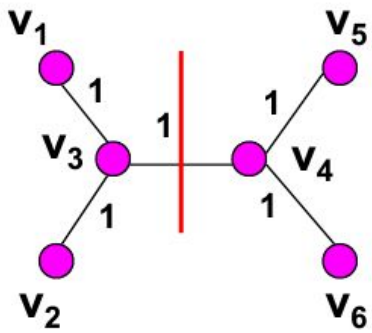
If graph is unweighted (or has the same edge weight)



Cut = 1

Ratio cut = 1/1 + 1/5 = 1.2

Normalized cut = 1/1 + 1/9
= 1.11



Cut = 2

Ratio cut = 1/3 + 1/3 = 0.67

Normalized cut = 1/5 + 1/5
= 0.2

Algorithm for Cut-based Graph Partitioning

- How to minimize the objective function?
  - use a heuristic (greedy) approach to do this
  - using ideas from spectral graph theory

Spectral Clustering

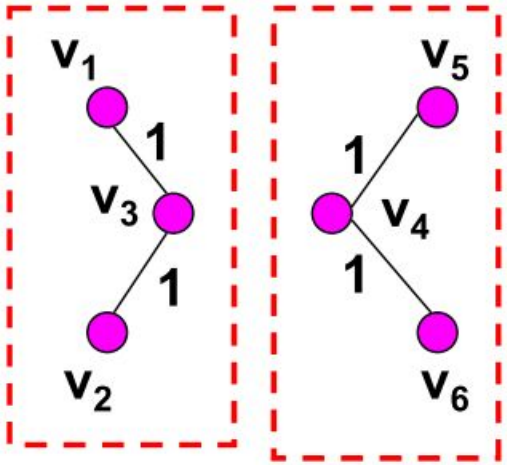
- Spectral properties of a graph
  - Spectral properties: eigenvalues/eigenvectors of the adjacency matrix can be used to represent a graph
- There exists a relationship between spectral properties of a graph and the graph partitioning problem

1. Start with a similarity/adjacency matrix, W, of a graph

2. Define a diagonal matrix D

$$D_{ij} = \begin{cases} \sum_{k=1}^{n} w_{ik} & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$$

- If W is a binary 0/1 matrix, then D_{ii} represents the degree of node i



Two clusters

$$W = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

Two block-diagonal matrices

$$D = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

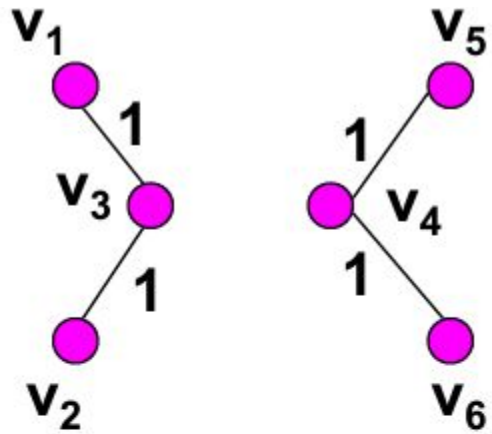
3. Laplacian, L= D-W

$$W = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

$$D = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$L = \begin{bmatrix} 1 & 0 & -1 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 \\ -1 & -1 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & -1 & -1 \\ 0 & 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & -1 & 0 & 1 \end{bmatrix}$$

L = (D –W) is a symmetric matrix
L is a positive semi-definite matrix
all eigenvalues of L are >= 0



If we cluster the data using only the first 2 eigenvectors, we get the two desired clusters

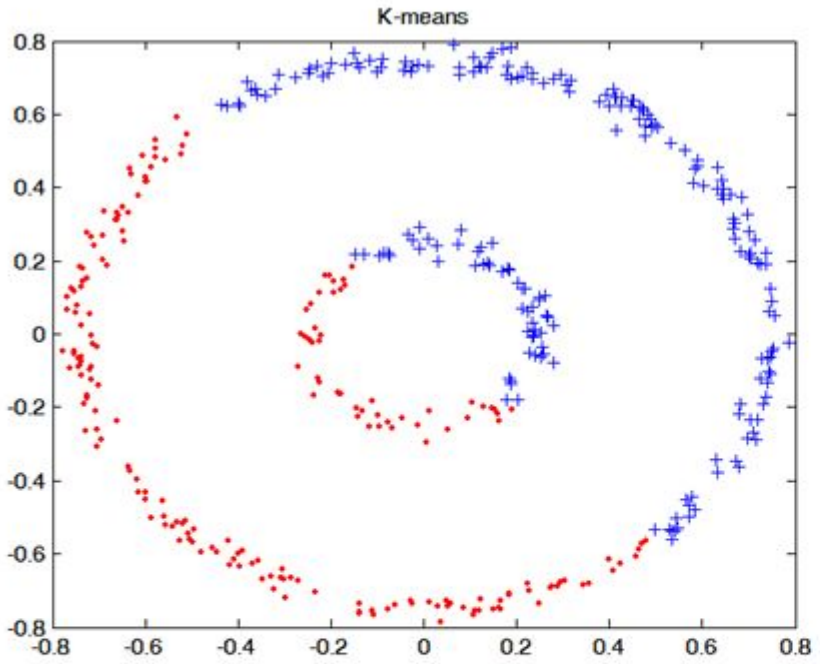
Eigenvalues of L:

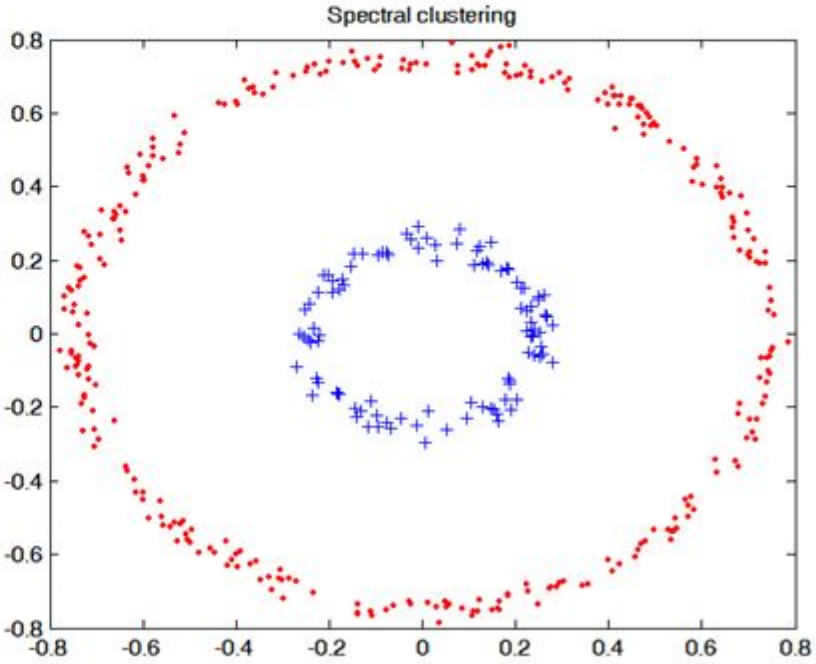
$$\Lambda = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 3 \end{bmatrix}$$

Eigenvectors of L:

$$V = \begin{bmatrix} 0.58 & 0 & 0.71 & 0 & 0 & -0.41 \\ 0.58 & 0 & -0.71 & 0 & 0 & -0.41 \\ 0.58 & 0 & 0 & 0 & 0 & 0.81 \\ 0 & -0.58 & 0 & 0 & -0.82 & 0 \\ 0 & -0.58 & 0 & -0.71 & 0.41 & 0 \\ 0 & -0.58 & 0 & 0.71 & 0.41 & 0 \end{bmatrix}$$

- Spectral properties of a graph (i.e., eigenvalues and eigenvectors) contain information about clustering structure
- To find k-clusters, apply k-means or other algorithms to the first k eigenvectors of the graph Laplacian matrix





Spectral Clustering Algorithm

Consider a data set with N data points

1. Construct an $N \times N$ similarity matrix, W
2. Compute the $N \times N$ Laplacian matrix, $L = D - W$
3. Compute the k "smallest" eigenvectors of L
   a) Each eigenvector v_i is an $N \times 1$ column vector
   b) Create a matrix V containing eigenvectors $v_1, v_2, .., v_k$ as columns (you may exclude the first eigenvector)
4. Cluster the rows in V using k-means or other clustering algorithms into K clusters