Clustering
Lecture 6: Spectral Methods

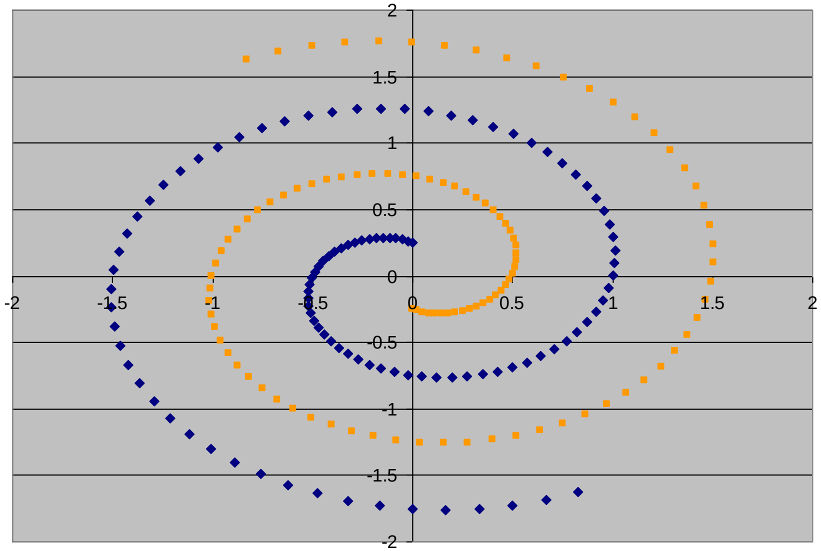
Jing Gao
SUNY Buffalo

Outline

- **Basics**
  - Motivation, definition, evaluation
- **Methods**
  - Partitional
  - Hierarchical
  - Density-based
  - Mixture model
  - Spectral methods
- **Advanced topics**
  - Clustering ensemble
  - Clustering in MapReduce
  - Semi-supervised clustering, subspace clustering, co-clustering, etc.

Motivation

- **Complex cluster shapes**
  - K-means performs poorly because it can only find spherical clusters
  - Density-based approaches are sensitive to parameters
- **Spectral approach**
  - Use similarity graphs to encode local neighborhood information
  - Data points are vertices of the graph
  - Connect points which are “close”

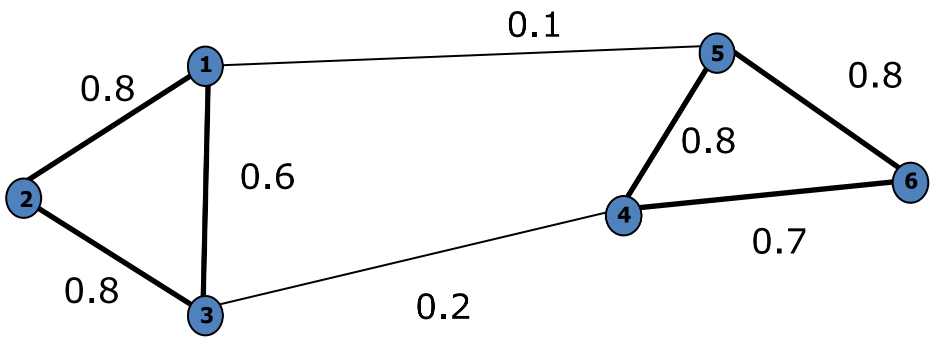


Similarity Graph

- Represent dataset as a weighted graph $G(V,E)$
- All vertices which can be reached from each other by a path form a connected component
- Only one connected component in the graph—The graph is fully connected

$V=\{x_i\}$ Set of n vertices representing data points

$E=\{W_{ij}\}$ Set of weighted edges indicating pair-wise similarity between points



Graph Construction

- ***ε*-neighborhood graph**
  - Identify a threshold value, *ε*, and include edges if the affinity between two points is greater than *ε*
- ***k*-nearest neighbors**
  - Insert edges between a node and its *k*-nearest neighbors
  - Each node will be connected to (at least) *k* nodes
- **Fully connected**
  - Insert an edge between every pair of nodes
  - Weight of the edge represents similarity
  - Gaussian kernel:

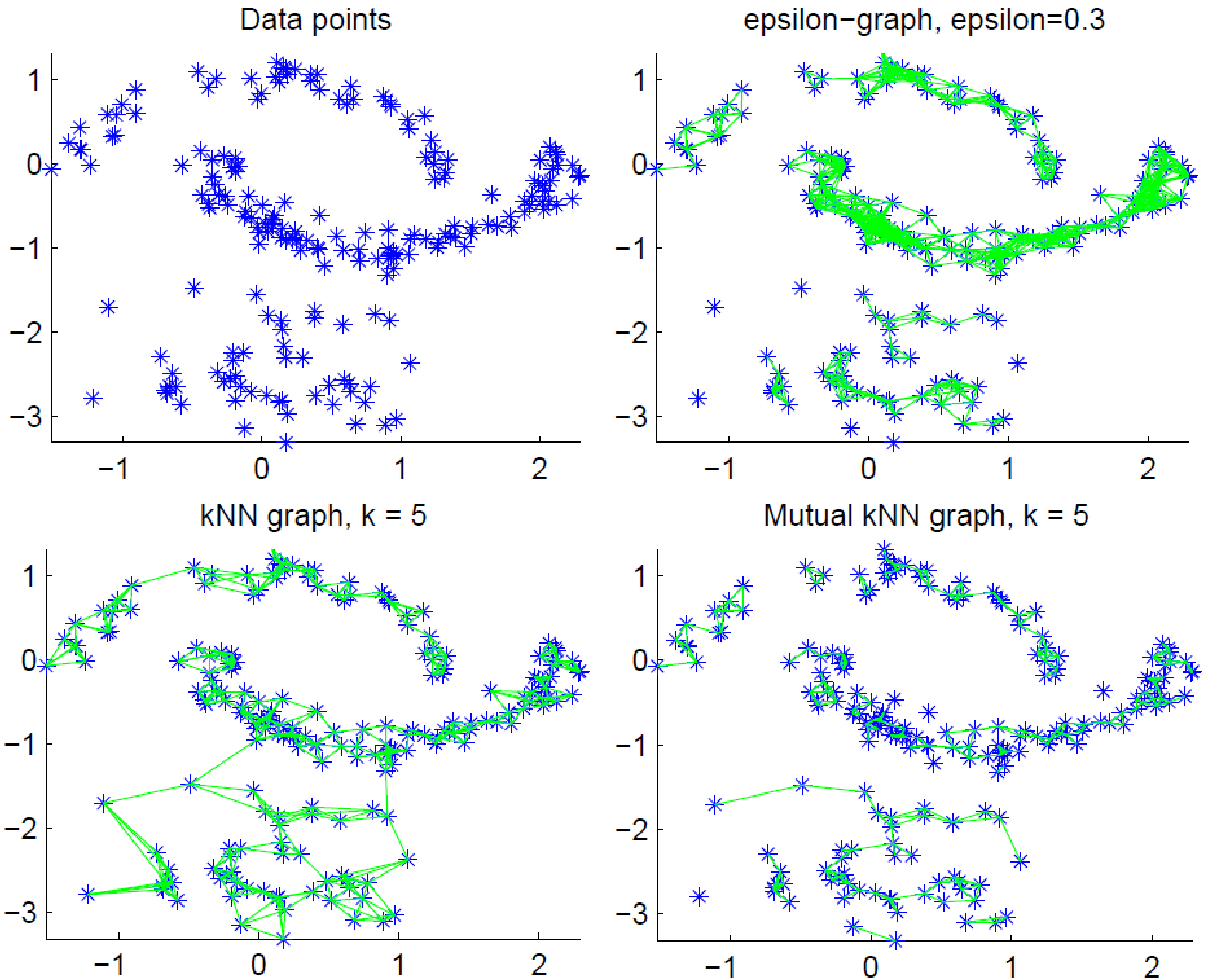
$$w_{ij} = \exp(-\| x_i - x_j \|^2 / \sigma^2)$$

ε-neighborhood Graph

- ***ε*-neighborhood**
  - Compute pairwise distance between any two objects
  - Connect each point to all other points which have distance smaller than a threshold ε
- **Weighted or unweighted**
  - Unweighted—There is an edge if one point belongs to the ε–neighborhood of another point
  - Weighted—Transform distance to similarity and use similarity as edge weights

*k*NN Graph

- **Directed graph**
  - Connect each point to its *k* nearest neighbors
- **kNN graph**
  - Undirected graph
  - An edge between x_i and x_j : There's an edge from x_i to x_j OR from x_j to x_i in the directed graph
- **Mutual kNN graph**
  - Undirected graph
  - Edge set is a subset of that in the kNN graph
  - An edge between x_i and x_j : There's an edge from x_i to x_j AND from x_j to x_i in the directed graph

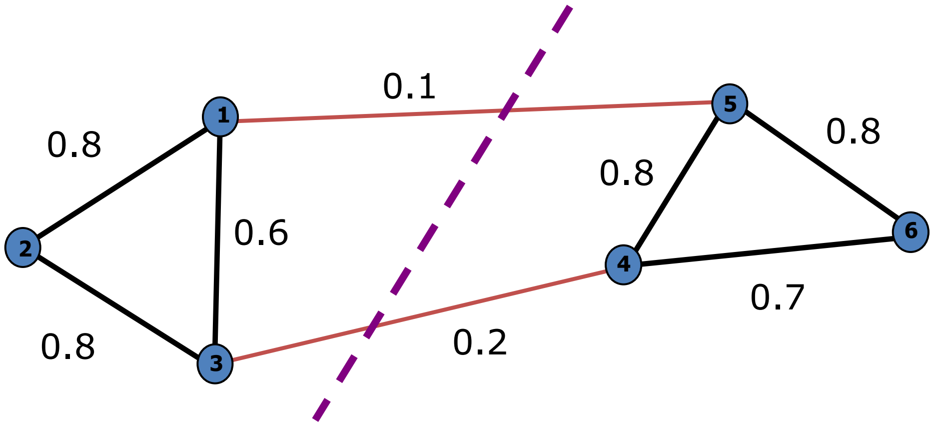

Data points
epsilon−graph, epsilon=0.3
kNN graph, k = 5
Mutual kNN graph, k = 5
1
0
−1
−2
−3
−1
0
1
2

Clustering Objective

Traditional definition of a "good" clustering

- Points assigned to same cluster should be highly similar
- Points assigned to different clusters should be highly dissimilar

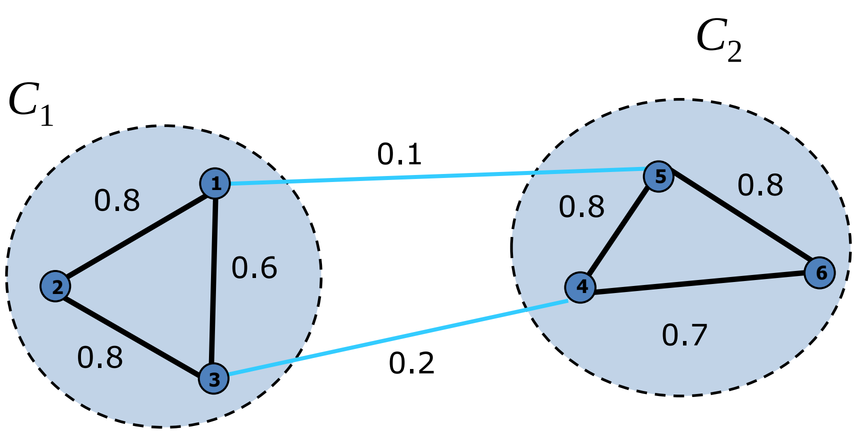
Apply this objective to our graph representation



Minimize weight of between-group connections

Graph Cuts

- Express clustering objective as a function of the ***edge cut*** of the partition
- ***Cut****:* Sum of weights of edges with only one vertex in each group
- We wants to find the ***minimal cut*** between groups



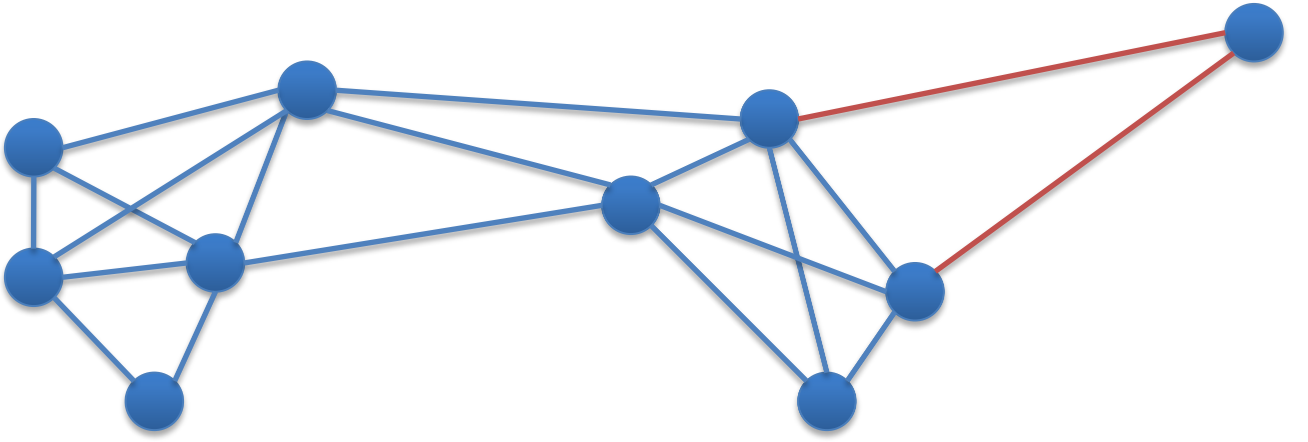
$$cut(C_1, C_2) = \sum_{i \in C_1,\, j \in C_2} w_{ij}$$

$$cut(C_1, C_2) = 0.3$$

Bi-partitional Cuts

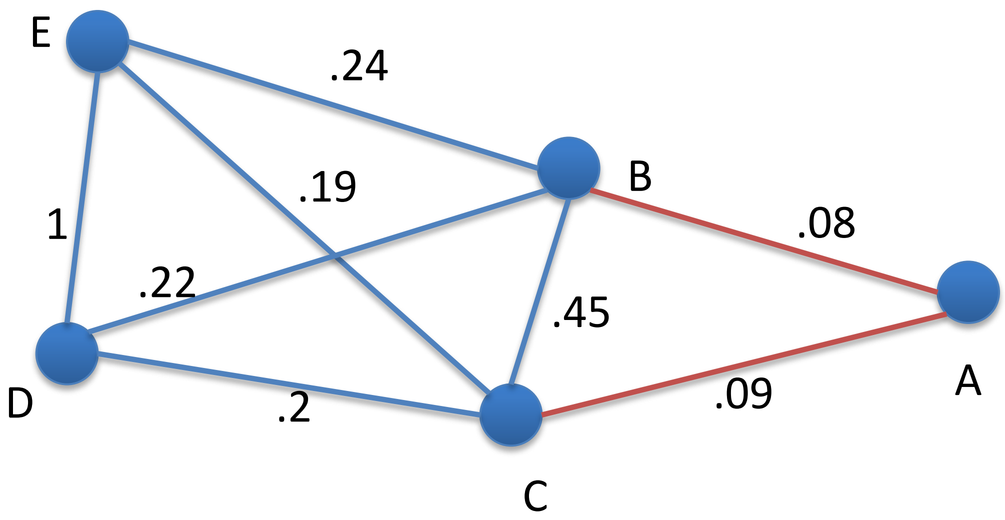
- Minimum (bi-partitional) cut

$$\min Cut(C_1, C_2) = \sum_{i \in C_1} \sum_{j \in C_2} w_{ij}$$



Example

- Minimum Cut

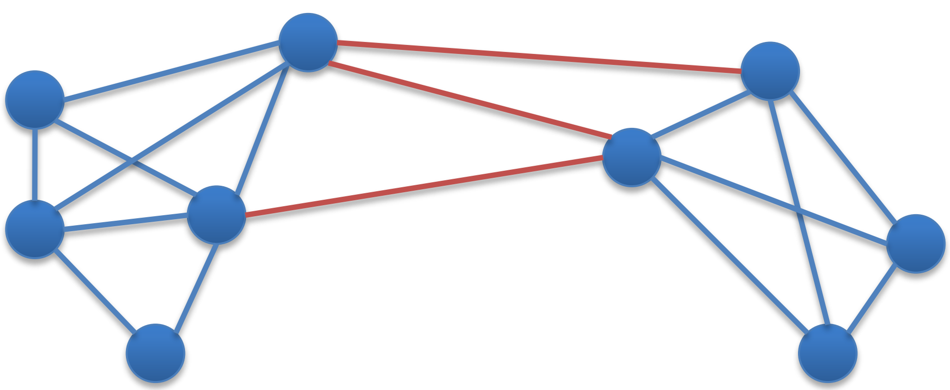


$$Cut(BCDE, A) = 0.17$$

Normalized Cuts

- Minimal (bipartitional) normalized cut

$$\min \frac{Cut(C_1, C_2)}{Vol(C_1)} + \frac{Cut(C_1, C_2)}{Vol(C_2)} = \min \left(\frac{1}{Vol(C_1)} + \frac{1}{Vol(C_2)} \right) Cut(C_1, C_2)$$

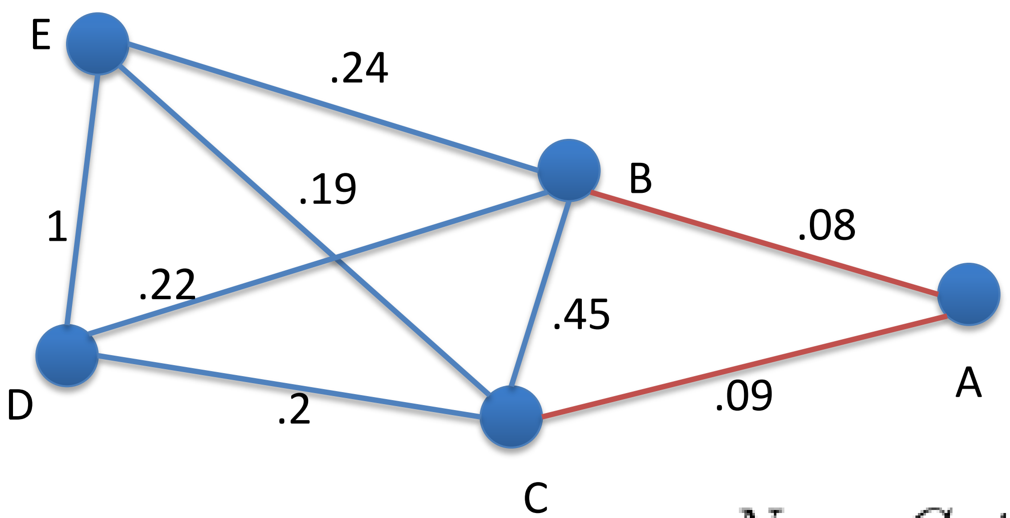


$$Vol(C) = \sum_{i \in C, j \in V} w_{ij}$$

Example

- Normalized Minimum Cut

$$NormCut(C_1, C_2) = \frac{Cut(C_1, C_2)}{Vol(C_1)} + \frac{Cut(C_1, C_2)}{Vol(C_2)}$$

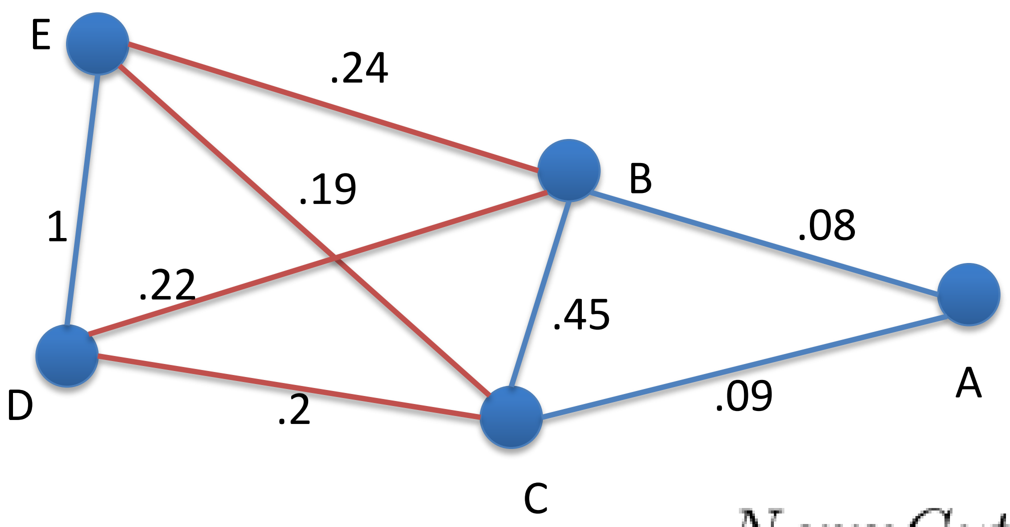


$$NormCut(BCDE, A) = 1.067$$

Example

- Normalized Minimum Cut

$$NormCut(C_1, C_2) = \frac{Cut(C_1, C_2)}{Vol(C_1)} + \frac{Cut(C_1, C_2)}{Vol(C_2)}$$



$$NormCut(BCDE, A) = 1.067$$

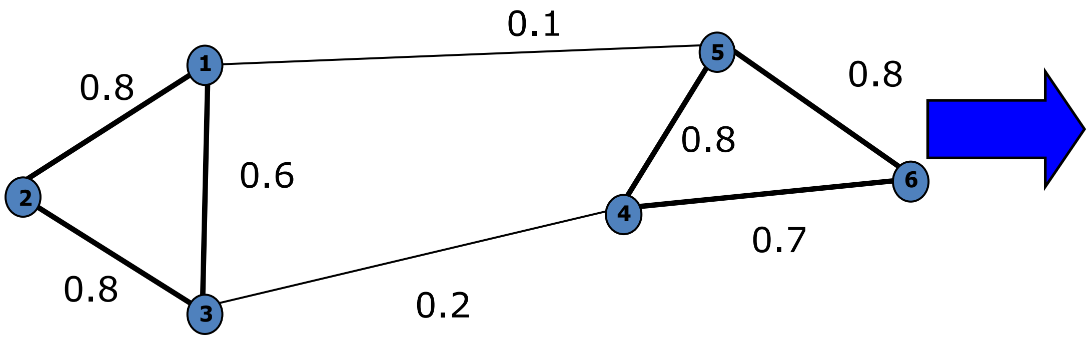
$$NormCut(ABC, DE) = 1.038$$

Problem

- Identifying a minimum cut is NP-hard
- There are efficient approximations using linear algebra
- Based on the Laplacian Matrix, or **graph Laplacian**

Matrix Representations

- **Similarity matrix (W)**
  - $n \times n$ matrix
  - $W = [w_{ij}]$: edge weight between vertex x_i and x_j

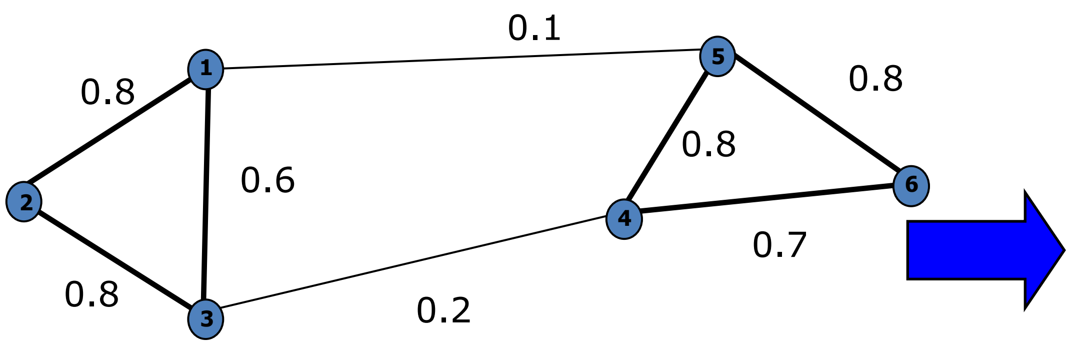


	x_1	x_2	x_3	x_4	x_5	x_6
x_1	0	0.8	0.6	0	0.1	0
x_2	0.8	0	0.8	0	0	0
x_3	0.6	0.8	0	0.2	0	0
x_4	0	0	0.2	0	0.8	0.7
x_5	0.1	0	0	0.8	0	0.8
x_6	0	0	0	0.7	0.8	0

- **Important properties**
  - Symmetric matrix

Matrix Representations

- **Degree matrix (D)**
  - $n \times n$ diagonal matrix
  - $D(i,i) = \sum_j w_{ij}$: total weight of edges incident to vertex x_i



	x_1	x_2	x_3	x_4	x_5	x_6
x_1	1.5	0	0	0	0	0
x_2	0	1.6	0	0	0	0
x_3	0	0	1.6	0	0	0
x_4	0	0	0	1.7	0	0
x_5	0	0	0	0	1.7	0
x_6	0	0	0	0	0	1.5

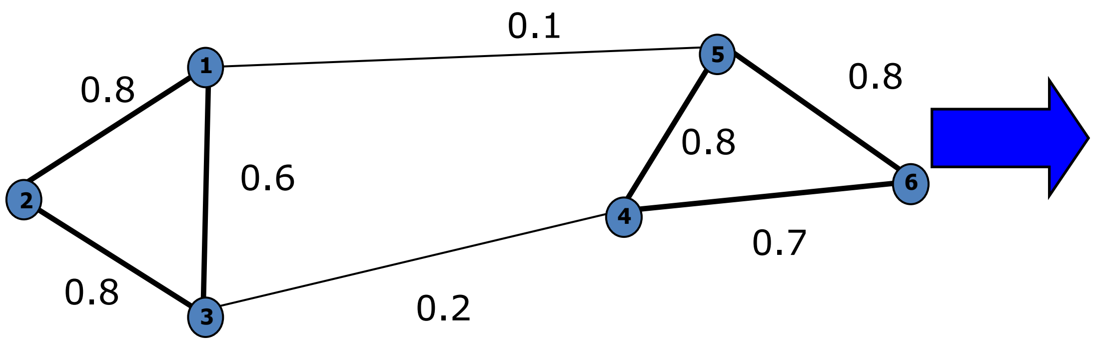
- **Used to**
  - Normalize adjacency matrix

Matrix Representations

- **Laplacian matrix (L)**

$$L = D - W$$

  - n x n symmetric matrix



	x_1	x_2	x_3	x_4	x_5	x_6
x_1	1.5	-0.8	-0.6	0	-0.1	0
x_2	-0.8	1.6	-0.8	0	0	0
x_3	-0.6	-0.8	1.6	-0.2	0	0
x_4	0	0	-0.2	1.7	-0.8	-0.7
x_5	-0.1	0	0	-0.8	1.7	-0.8
x_6	0	0	0	-0.7	-0.8	1.5

- **Important properties**
  - Eigenvalues are non-negative real numbers
  - Eigenvectors are real and orthogonal
  - Eigenvalues and eigenvectors provide an insight into the connectivity of the graph…

Find An Optimal Min-Cut (Hall'70, Fiedler'73)

- Express a bi-partition (C_1,C_2) as a vector

$$f_i = \begin{cases} 1 & if \;\; x_i \in C_1 \\ -1 & if \;\; x_i \in C_2 \end{cases}$$

- We can minimise the cut of the partition by finding a non-trivial vector f that minimizes the function

$$g(f) = \sum_{i,j \in V} w_{ij}(f_i - f_j)^2 = 2 f^T L \; f$$

Laplacian matrix

Why does this work?

$$L = D - W$$

$$\begin{aligned} f^T L f &= f^T D f - f^T W f \\ &= \sum_i d_i f_i^2 - \sum_{ij} f_i f_j w_{ij} \\ &= \frac{1}{2}\left(\sum_i \left(\sum_j w_{ij}\right) f_i^2 - 2\sum_{ij} f_i f_j w_{ij} + \sum_j \left(\sum_i w_{ij}\right) f_j^2\right) \\ &= \frac{1}{2}\sum_{ij} w_{ij}(f_i - f_j)^2 \end{aligned}$$

--Clustering objective function

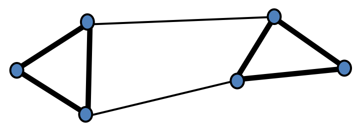
- $Lf = \lambda f \quad f^T L f = f^T \lambda f = \lambda f^T f = \lambda$
- If we let *f* be eigen vectors of *L*, then the corresponding eigen value is the value of the clustering objective function

Optimal Min-Cut

- The Laplacian matrix L is semi positive definite
- The Rayleigh Theorem shows:
  - The minimum value for $g(f)$ is given by the 2nd smallest eigenvalue of the Laplacian L
  - The optimal solution for f is given by the corresponding eigenvector λ_2, referred as the Fiedler Vector

Spectral Bi-partitioning Algorithm

1. Pre-processing
   - Build Laplacian matrix L of the graph



	x_1	x_2	x_3	x_4	x_5	x_6
x_1	1.5	-0.8	-0.6	0	-0.1	0
x_2	-0.8	1.6	-0.8	0	0	0
x_3	-0.6	-0.8	1.6	-0.2	0	0
x_4	0	0	-0.2	1.7	-0.8	-0.7
x_5	-0.1	0	0	-0.8	1.7	-0.8
x_6	0	0	0	-0.7	-0.8	1.5

2. Decomposition
   - Find eigenvectors X and eigenvalues Λ of the matrix L

$\Lambda =$

0.0
0.4
2.2
2.3
2.5
3.0

$X =$

0.4	0.2	0.1	0.4	-0.2	-0.9
0.4	0.2	0.1	-0.	0.4	0.3
0.4	0.2	-0.2	0.0	-0.2	0.6
0.4	-0.4	0.9	0.2	-0.4	-0.6
0.4	-0.7	-0.4	-0.8	-0.6	-0.2
0.4	-0.7	-0.2	0.5	0.8	0.9

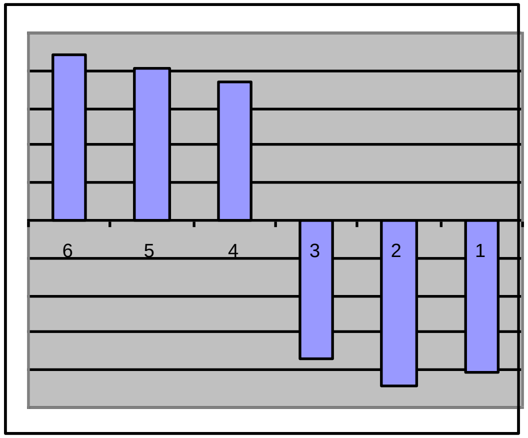
   - Map vertices to corresponding components of λ_2

x_1	0.2
x_2	0.2
x_3	0.2
x_4	-0.4
x_5	-0.7
x_6	-0.7

Spectral Bi-partitioning Algorithm

The matrix which represents the eigenvector of the Laplacian (the eigenvector matched to the corresponded eigenvalues with increasing order)

0.41	-0.41	0.65-	0.31-	0.38-	0.11
0.41	-0.44	0.01	0.30	0.71	0.22
0.41	-0.37	0.64	0.04	0.39-	0.37-
0.41	0.37	0.34	0.45-	0.00	0.61
0.41	0.41	0.17-	0.30-	0.35	0.65-
0.41	0.45	0.18-	0.72	0.29-	0.09



Spectral Bi-partitioning

- Grouping
  - Sort components of reduced 1-dimensional vector
  - Identify clusters by splitting the sorted vector in two (above zero, below zero)

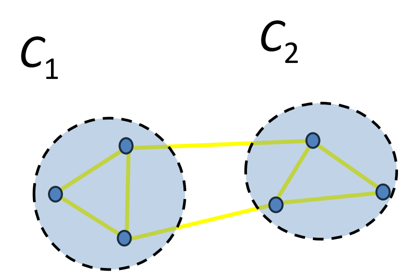
x_1	0.2
x_2	0.2
x_3	0.2
x_4	-0.4
x_5	-0.7
x_6	-0.7

Split at 0

- Cluster C_1: Positive points
- Cluster C_2: Negative points

x_1	0.2
x_2	0.2
x_3	0.2

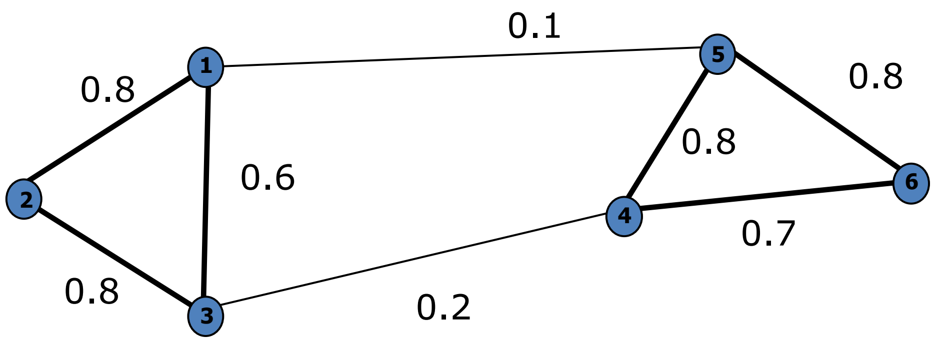
x_4	-0.4
x_5	-0.7
x_6	-0.7



Normalized Laplacian

- **Laplacian matrix (*L*)**

$$L = D^{-1}(D - W)$$
$$L = D^{-0.5}(D - W)D^{-0.5}$$

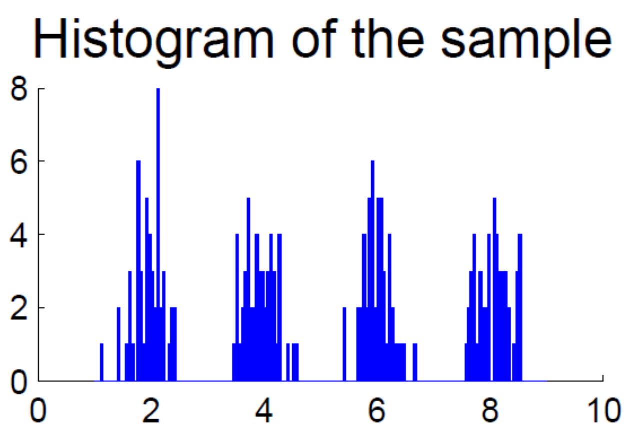


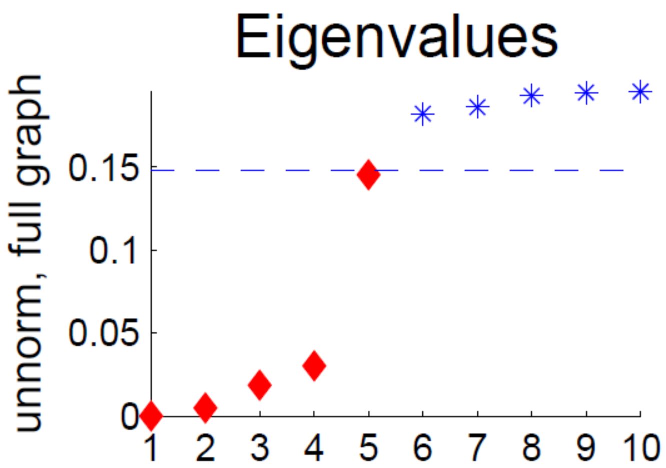
1.00	-0.52	-0.39	0.00	-0.06	0.00
-0.52	1.00	-0.50	0.00	0.00	0.00
-0.39	-0.50	1.00	-0.12	0.00	0.00
0.00	0.00	-0.12	1.00	0.47-	0.44-
-0.06	0.00	0.00	-0.47	1.00	0.50-
0.00	0.00	0.00	0.44-	0.50-	1.00

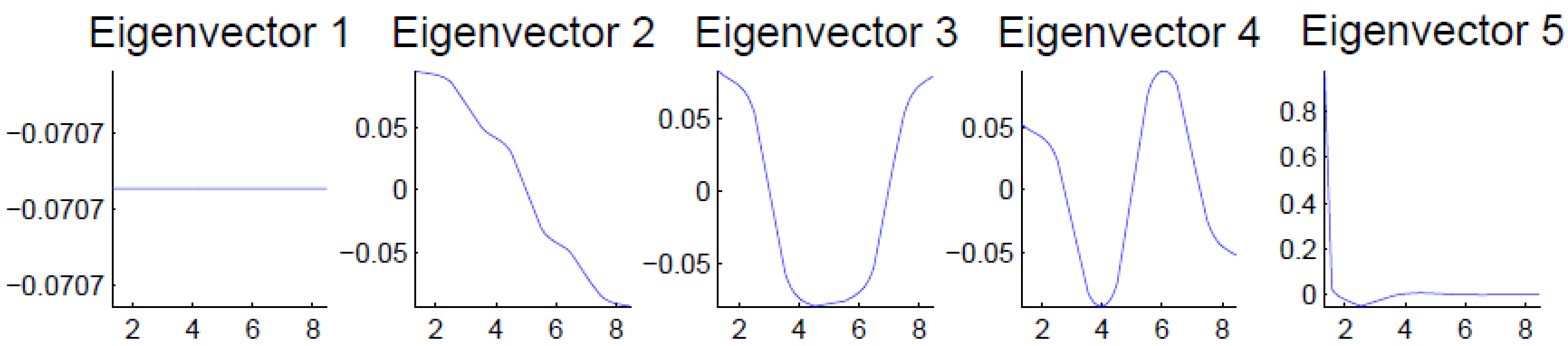
K-Way Spectral Clustering

- How do we partition a graph into k clusters?
  1. **Recursive bi-partitioning** (Hagen et al.,'91)
     - Recursively apply bi-partitioning algorithm in a hierarchical divisive manner.
     - Disadvantages: Inefficient, unstable
  2. **Cluster multiple eigenvectors** (Shi & Malik,'00)
     - Build a reduced space from multiple eigenvectors.
     - Commonly used in recent papers
     - A preferable approach

Eigenvectors & Eigenvalues







K-way Spectral Clustering Algorithm

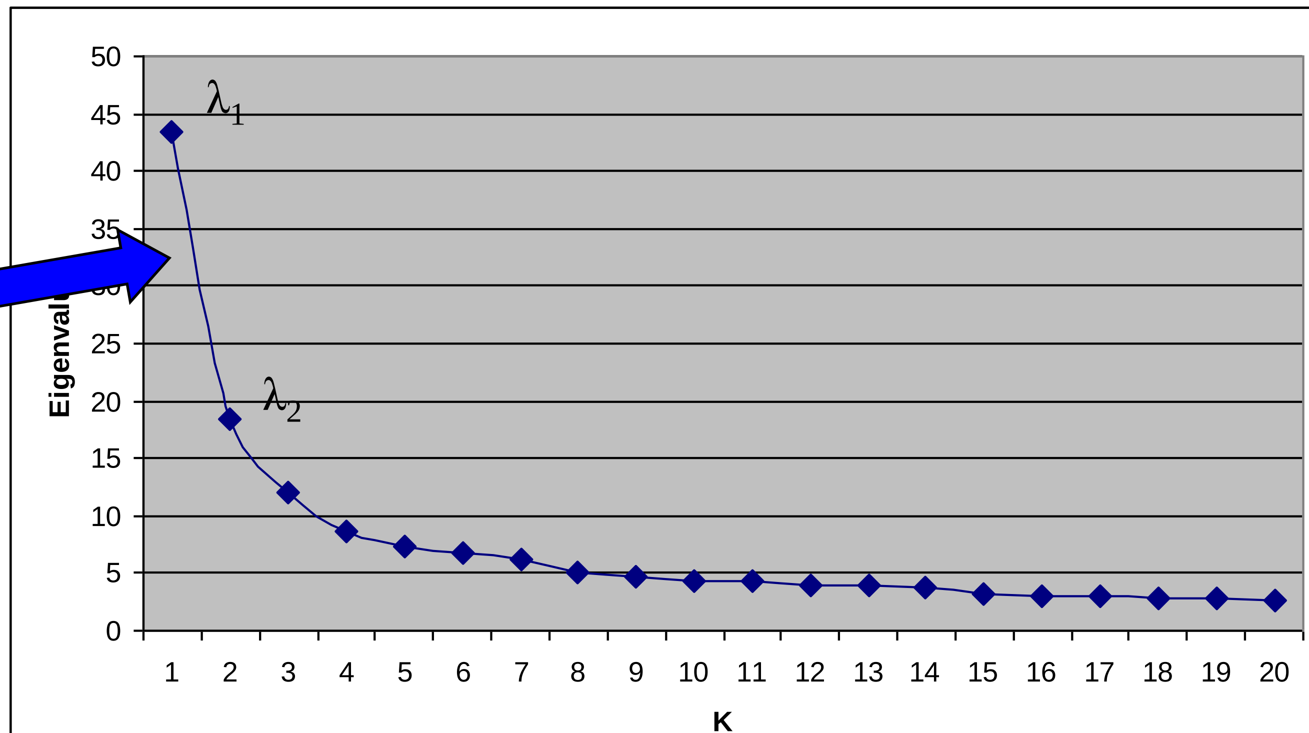
- **Pre-processing**
  - Compute Laplacian matrix *L*
- **Decomposition**
  - Find the eigenvalues and eigenvectors of *L*
  - Build embedded space from the eigenvectors corresponding to the *k* smallest eigenvalues
- **Clustering**
  - Apply *k*-means to the reduced *n* x *k* space to produce *k* clusters

How to select *k*?

- *Eigengap*: the difference between two consecutive eigenvalues
- Most stable clustering is generally given by the value k that maximizes the expression $$\Delta_k = |\lambda_k - \lambda_{k-1}|$$

$$\max \Delta_k = |\lambda_2 - \lambda_1|$$

$\Rightarrow$ Choose *k*=2



Take-away Message

- Clustering formulated as graph cut problem
- How min-cut can be solved by eigen decomposition of Laplacian matrix
- Bipartition and multi-partition spectral clustering procedure