# Chapter 8-2:

# Community Detection

Jilles Vreeken

IRDM '15/16

3 Dec 2015

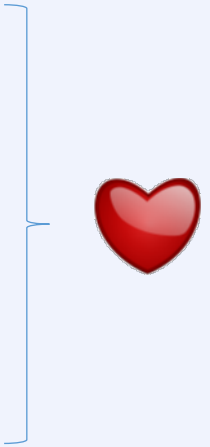
UNIVERSITÄT DES SAARLANDES

CLUSTER OF EXCELLENCE

max planck institut informatik

# IRDM Chapter 8, overview

1. The basics
2. Properties of Graphs
3. Frequent Subgraphs
4. Graph Clustering



You'll find this covered in:
Aggarwal, Ch. 17, 19
Zaki & Meira, Ch. 4, 11, 16

# IRDM Chapter 8, today

1. The basics
2. Properties of Graphs
3. Frequent Subgraphs
4. Community Detection

You'll find this covered in:
Aggarwal, Ch. 17, 19
Zaki & Meira, Ch. 4, 11, 16

# Chapter 7.4:
# Community Detection

Aggarwal Ch. 19.3, 17.5
Zaki & Meira Ch. 16

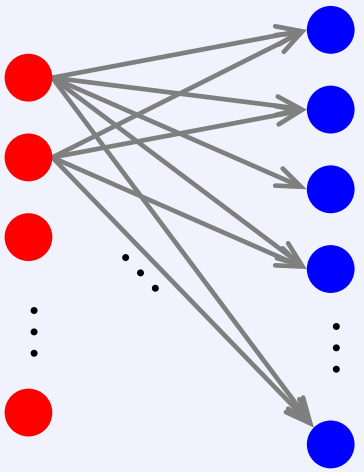
# Chapter 7.4.1:
# Detecting Small Communities

# Trawling

Searching for small communities in the Web graph

What is the signature of a community in a Web graph?

- **intuition:** Many people all talking about the same things



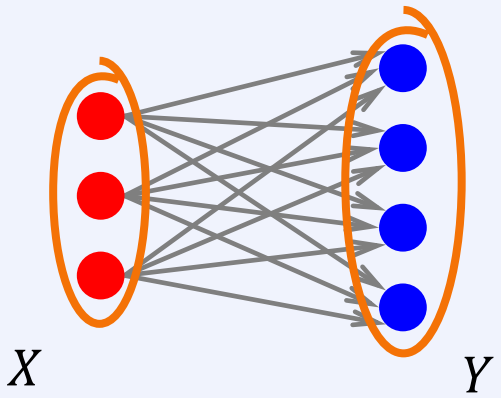
Dense 2-layer graph

**Use this to define "topics":**
What the same people on the left talk about on the right

# Searching for small communities

A more well-defined problem:

- enumerate complete bipartite subgraphs $K_{\alpha,\beta}$ where $K_{\alpha,\beta}$ has $s$ nodes on the "left" and every such node in $s$ links to the same set of $t$ nodes on the "right"




$K_{3,4}$

$$|X| = \alpha = 3$$
$$|Y| = \beta = 4$$

**Fully connected**

# Frequent itemset mining

Recall market basket analysis.

- market: universe $U$ of $n$ items
- baskets: $m$ transctions, subsets of $U$: $t_1, t_2, \ldots, t_m \subseteq U$ where each $t_i$ is a set of items one person bought
- support: frequency threshold $\sigma$

Goal:

- find all subsets $X \subseteq U$ s.t. $X \subseteq t_i$ of at least $\sigma$ sets $t_i \in \boldsymbol{D}$

What's the connection between itemsets and complete bipartite graphs?

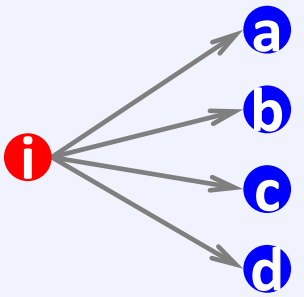
# From itemsets to bipartite $K_{\alpha,\beta}$

Frequent itemsets = complete bipartite graphs!

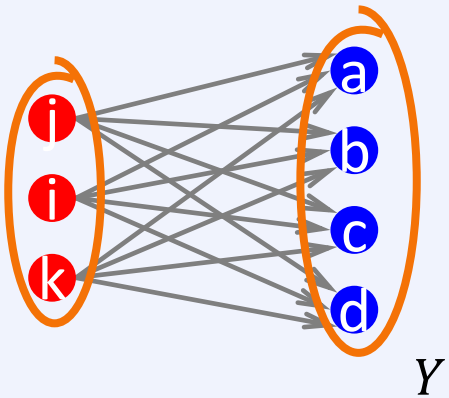
How?

- view each node $i$ as a set $t_i$ of the nodes $i$ points to
  $K_{\alpha,\beta}$ = a set $Y$ of size $\beta$ that occurs in $\alpha$ sets $t_i$

- looking for $K_{\alpha,\beta}$ →
  set frequency threshold to $\alpha$ and look at layer $\beta$ , find all frequent sets of size $t$




$t_i = \{a, b, c, d\}$

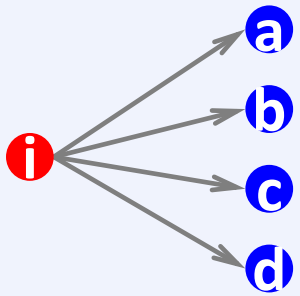



$\alpha$ … minimum support ($|X| = \alpha$)

$\beta$ … itemset size ($|Y| = \beta$)

# From itemsets to bipartite $K_{\alpha,\beta}$

1) View each node $i$ as a set $t_i$ of nodes $i$ points to




$t_i = \{a, b, c, d\}$

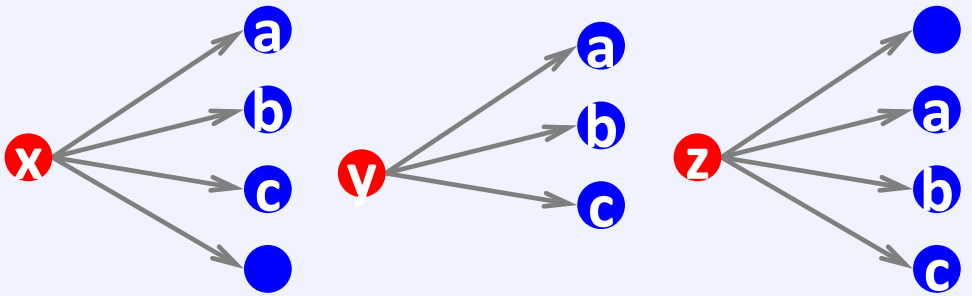
2) Find frequent itemsets:

$\alpha$ … minimum support

$\beta$ … itemset size

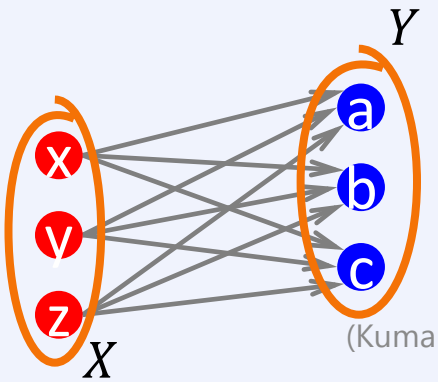
3) Say we find a **frequent itemset** $X = \{a, b, c\}$ of supp $\alpha$

This means, there are $\alpha$ nodes that link to all of $\{a, b, c\}$:



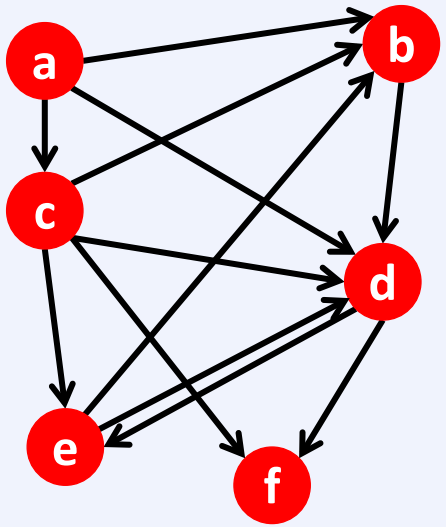

4) **We found $K_{\alpha,\beta}$!**

$K_{\alpha,\beta}$ = a set $Y$ of size $\beta$ that occurs in $\alpha$ sets $t_i$




(Kumar et al '99)

# Example



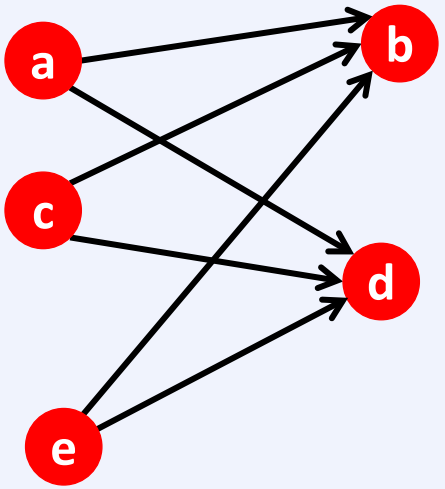

**Support threshold** $\alpha = \sigma = 2$

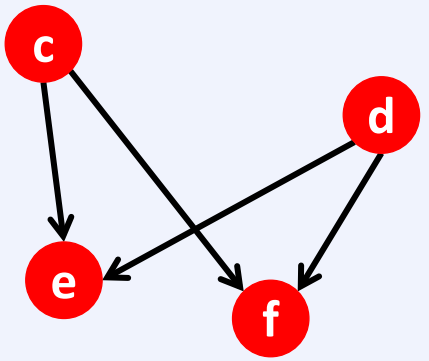
- $\{b, d\}$: support 3
- $\{e, f\}$: support 2

- i.e. we found 2 bipartite subgraphs:

**Itemsets:**

$a = \{b, c, d\}$
$b = \{d\}$
$c = \{b, d, e, f\}$
$d = \{e, f\}$
$e = \{b, d\}$
$f = \{\}$



# Chapter 7.4.2: Community Detection by Graph Clustering

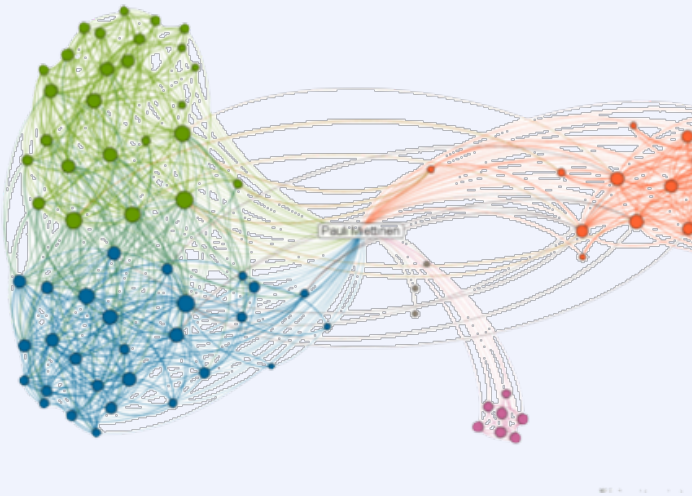
Aggarwal Ch. 17.5, 19.3

Zaki & Meira Ch. 16

# Where do graphs come from?

We can have data in graph form

- e.g. the clusters of our social networks



Or, we map existing data to a graph

- data points become vertices
- add an edge if two data points are similar
  - edge weights can also tell about similarity

# Similarity and adjacency matrices

A **similarity** matrix is an $n$-by- $n$ non-negative, symmetric matrix

- the opposite of the distance matrix

Recall that a weighted adjacency matrix is an $n$-by-$n$ non-negative, symmetric matrix

- for weighted, undirected graphs

So, we can think **every similarity matrix** as an adjacency matrix of some weighted, undirected graph

- this graph will be complete (a clique)

Further, we can use **any similarity measure** between two points as an edge weight

# Getting non-complete graphs

Using complete graphs can be a waste of resources

- for clustering, we don't really care about very dissimilar pairs

We can remove edges between dissimilar vertices

- zero weight

Or, we adjust the weights to diminish dissimilar points

- the **Gaussian kernel** is popular for this

$$w_{ij} = \exp\left\{-\frac{|x_i - x_j|^2}{2\sigma^2}\right\}$$

# Getting non-complete graphs (2)

How to decide when vertices are too dissimilar?

In **$\epsilon$-neighbour graphs** we add an edge between two vertices that are within distance $\epsilon$ to each other

- usually the resulting graph is considered unweighted as all weights would be roughly similar

In **$k$-nearest neighbour** graphs we connect two vertices if one is within the $k$ nearest neighbours of the other

- in **mutual $k$-nearest neighbour graph** we only connect two vertices if they're both in each other's $k$ nearest neighbours

# Which similarity graph?

With $\epsilon$-graphs choosing the parameter is hard

- **no single correct answer** if different clusters have different internal similarities

$k$-nearest neighbours can connect points with different similarities

- but far-away high density regions become unconnected

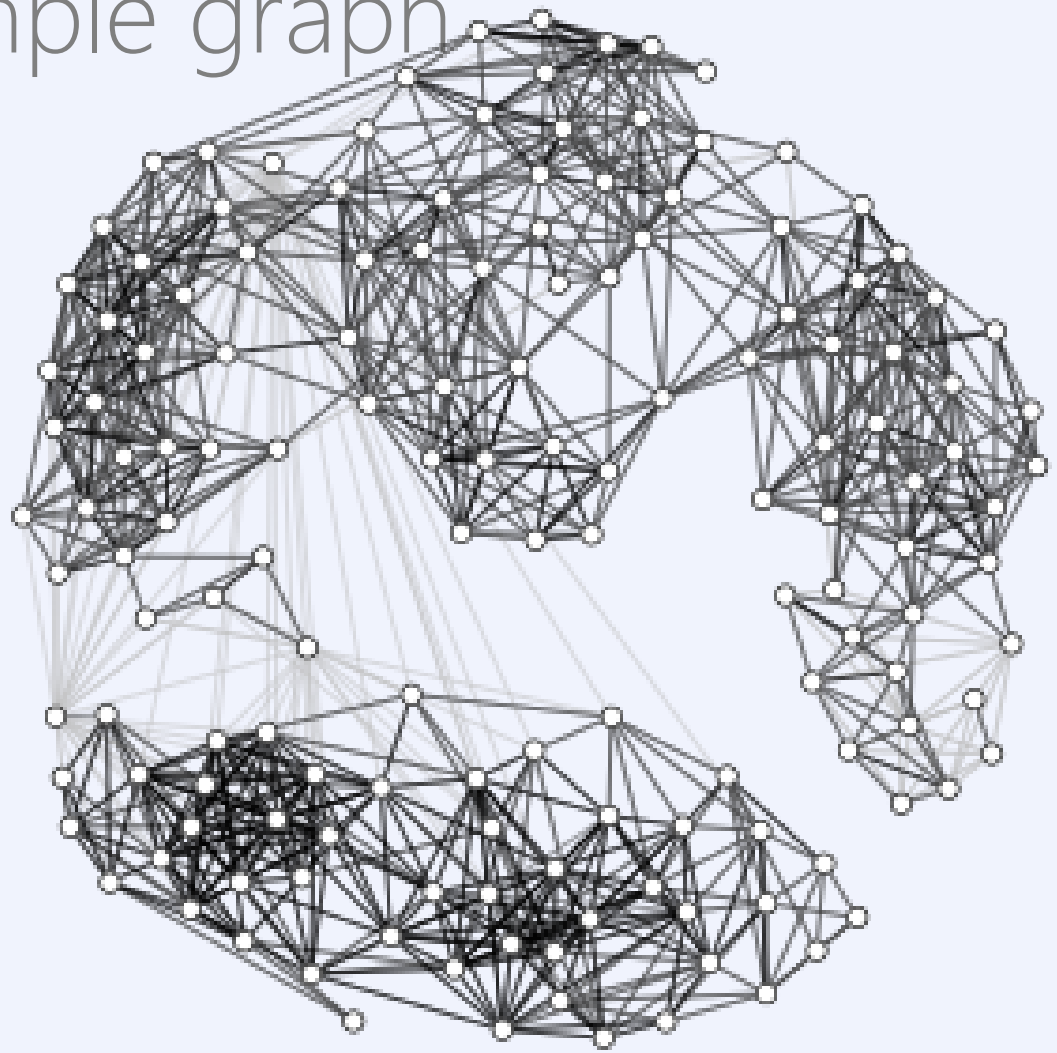
The mutual $k$-nearest neighbours is somewhat in between

- good for detecting clusters with different densities

General recommendation: start with $k$-NN

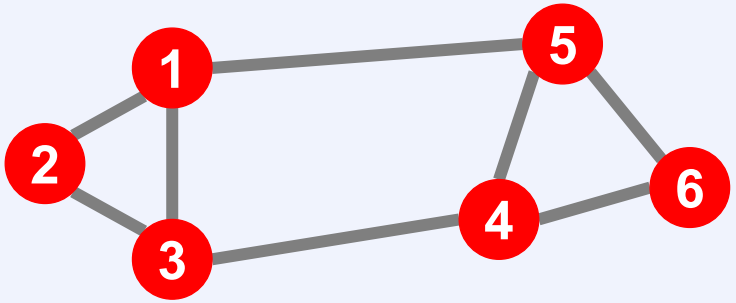
- others if data supports that

# Example graph



(Zaki & Meira, Fig 16.1)

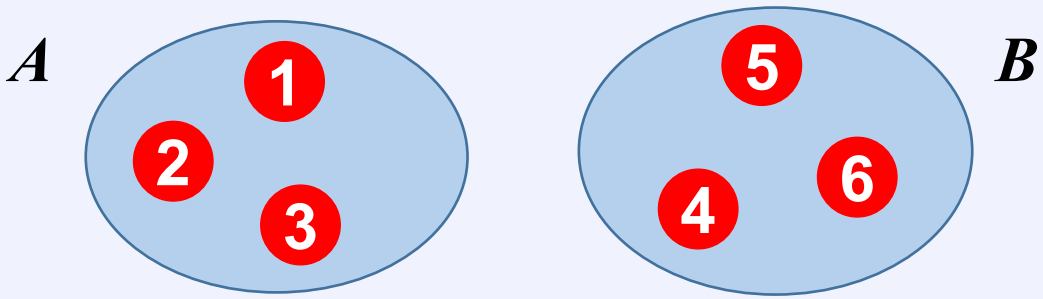
# Graph partitioning




Undirected graph

Bi-partitioning task:

- divide vertices into two disjoint groups



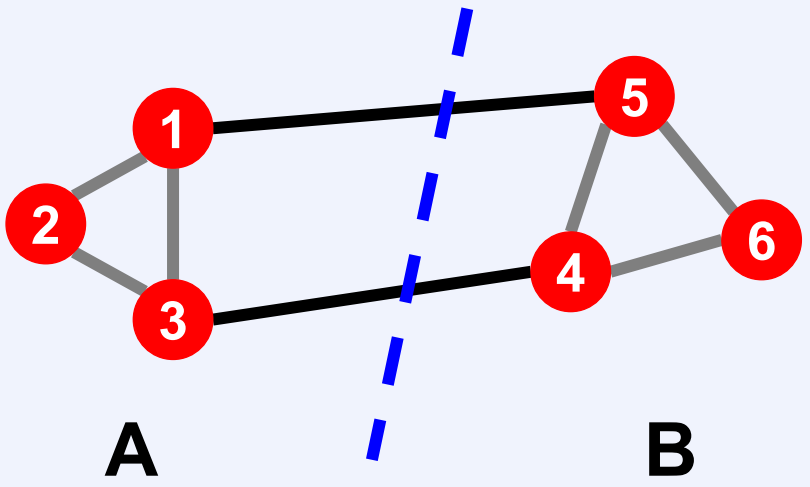

Questions:

- how can we define a "good partition of"?
- how can we efficiently identify such a partition?

# Graph partitioning

What makes a good partition?

- maximize the number of within-group connections
- minimize the number of between-group connections



# Clustering as Graph Cuts

A **cut** of a connected graph $G = (V, E)$ divides the set of vertices into two partitions $S$ and $V \setminus S$ and removes the edges between them

- cut can be expressed by giving the set $S$
- or by giving the cut set, i.e. edges with exactly one end in $S$,

$$F = \{(v, u) \in E : |\{v, u\} \cap S| = 1\}$$

A graph cut groups the vertices of a graph into two clusters

- subsequent cuts in the components give us a hierarchical clustering

A ***k*-way cut** cuts the graph into $k$ disjoint set of vertices $C_1, C_2, \ldots, C_k$ and removes the edges between them

# What is a good cut?

Not every cut will cut it

In **minimum cut** the goal is to find any set of vertices such that cutting them from the rest of the graph requires removing the least number of edges

- least sum of weights for weighted graphs
- the extension to multiway cuts is straightforward

The minimum cut can be found in polynomial time

- the max-flow min-cut theorem

But minimum cut isn't very good for clustering purposes

# Cuts that cut it

The minimum cut usually removes only one vertex

- not very appealing clustering
- we want to penalize the cut for imbalanced cluster sizes

In **ratio cut**, the goal is to minimize the ratio of the weight of the edges in the cut set and the size of the clusters $C_i$

- Let $W(A, B) = \sum_{i \in A, j \in B} w_{ij}$
  - wij is the weight of edge (i, j)

$$\text{RatioCut} = \sum_{i=1}^{k} \frac{W(C_i, V \setminus C_i)}{|C_i|}$$

# Cuts that cut it

The **volume** of a set of vertices $A$ is the weight of all edges connected to $A$

$$vol(A) = W(A, V) = \sum_{i \in A, j \in V} w_{ij}$$

In **normalized cut** we measure the size of $C_i$ by $vol(C_i)$ instead of $|C_i|$

$$\text{NormalisedCut} = \sum_{i=1}^{k} \frac{W(C_i, V \setminus C_i)}{vol(C_i)}$$

# Cuts that cut it

The **volume** of a set of vertices $A$ is the weight of all edges c

In norm
instead

Finding the optimal RatioCut or NormalisedCut is **NP**-hard

$$\text{NormalisedCut} = \sum_{i=1}^{k} \frac{W(C_i, V \setminus C_i)}{vol(C_i)}$$

# Spectral Graph Partitioning

- $\boldsymbol{A}$: adjacency matrix of undirected G
  - $\boldsymbol{A}_{ij} = 1$ if is an edge, else $0$
- $\boldsymbol{x}$ is a vector in $\mathbb{R}^n$ with components ('value groups')
  - think of it as a label/value of each node

What is the meaning of $\boldsymbol{A} \cdot \boldsymbol{x}$?

$$\begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}$$

$$y_i = \sum_j^n \boldsymbol{A}_{ij} x_j = \sum_{(i,j) \in E} \boldsymbol{x}_j$$

Entry $y_i$ is a sum of labels $x_j$ of neighbors of $i$

# What is the meaning of $\boldsymbol{Ax}$?

$j^{th}$ coordinate of $\boldsymbol{A} \cdot \boldsymbol{x}$

- sum of the $\boldsymbol{x}$-values of neighbors of $\boldsymbol{j}$
- make this a new value at node $\boldsymbol{j}$

$$\begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} = \lambda \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}$$

$$\boldsymbol{A} \cdot \boldsymbol{x} = \lambda \cdot \boldsymbol{x}$$

Spectral graph theory

- analyse the **spectrum** of the matrix
- the spectrum are the eigenvectors of a graph, ordered by the magnitude (strength) of their corresponding eigenvalues

$$\Lambda = \{\lambda_1, \lambda_2, \ldots, \lambda_n\} \text{ with}$$
$$\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$$

# Example: $d$-regular graph

Suppose all nodes in connected graph $G$ have degree $d$

What are some eigenvalues/vectors of $G$?

$$\boldsymbol{A} \cdot \boldsymbol{x} = \lambda \cdot \boldsymbol{x} \quad \text{What is } \lambda\text{? What is } \boldsymbol{x}\text{?}$$

- let's try: $\boldsymbol{x} = (1,1,\ldots,1)$
- then: $\boldsymbol{A} \cdot \boldsymbol{x} = (d,d,\ldots,d) = \lambda \cdot \boldsymbol{x}$. So: $\lambda = d$

We found eigenpair of $G$: $\boldsymbol{x} = (1,1,\ldots,1), \lambda = d$

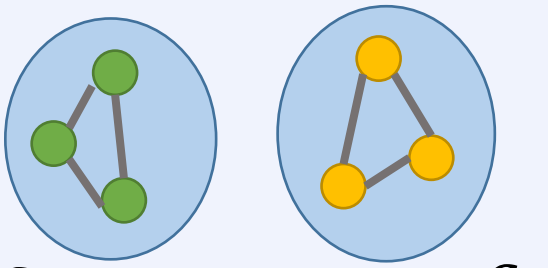
Remember the meaning of $\boldsymbol{y} = \boldsymbol{A} \cdot \boldsymbol{x}$:

$$y_i = \sum_j \boldsymbol{A}_{ij} x_j = \sum_{(i,j) \in E} \boldsymbol{x}_j$$

# Example: Graph of 2 components

What if $G$ is not connected?

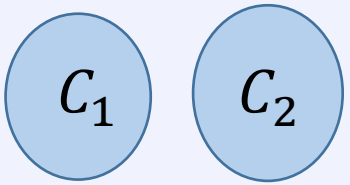
- $G$ has 2 components, each $d$-regular



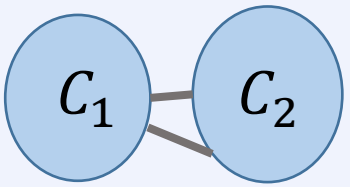

What are some eigenvectors?

- $\boldsymbol{x}$ = put all 1s on $\boldsymbol{C_1}$ and 0s on $\boldsymbol{C_2}$ or vice versa
  - $x' = (1, \dots, 1, 0, \dots, 0)$ then $\boldsymbol{A} \cdot x' = (\mathrm{d}, \dots, \mathrm{d}, 0, \dots, 0)$
  - $x'' = (0, \dots, 0, 1, \dots, 0)$ then $\boldsymbol{A} \cdot x'' = (0, \dots, 0, \mathrm{d}, \dots, \mathrm{d})$
  - and so, in both cases the corresponding $\lambda = d$

A bit of intuition:



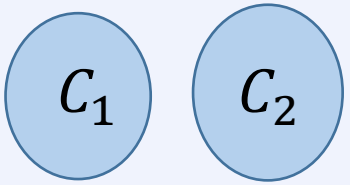

$\lambda_n = \lambda_{n-1}$



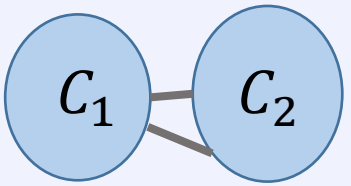

$\lambda_n - \lambda_{n-1} \approx 0$

2nd largest eigenvalue $\lambda_{n-1}$ now has value very close to $\lambda_n$

# More intuition




$\lambda_n = \lambda_{n-1}$




$\lambda_n - \lambda_{n-1} \approx 0$

2nd largest eigenvalue $\lambda_{n-1}$ now has value very close to $\lambda_n$

If the graph is connected (right) then we already know that $\boldsymbol{x}_n = (1, \ldots, 1)$ is an eigenvector

Since eigenvectors are orthogonal, the components of $\boldsymbol{x}_{n-1}$ sum to 0

- why? Because $\boldsymbol{x}_n \cdot \boldsymbol{x}_{n-1} = \sum_i \boldsymbol{x}_n[i] \cdot \boldsymbol{x}_{n-1}[i]$

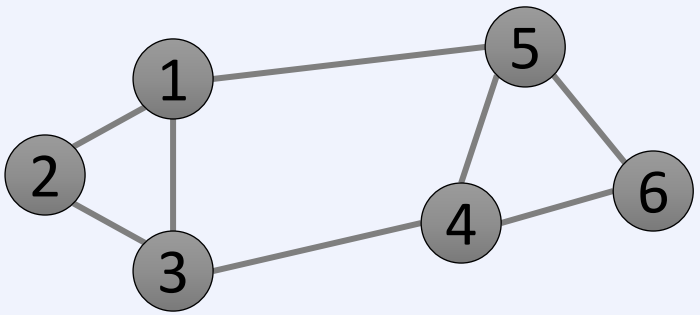
So, we can look at the eigenvectors of the $2^{nd}$ largest eigenvalue and declare nodes with positive label in $C_1$ and negative label in $C_2$

Still, lots to sort out.

# Matrix representations

The (weighted) adjacency matrix $\boldsymbol{A}$ has the weight of edge $(i, j)$ at position $\boldsymbol{a_{ij}}$

- $\boldsymbol{n}\times\boldsymbol{n}$ matrix
- $\boldsymbol{A} = [\boldsymbol{a_{ij}}], \boldsymbol{a_{ij}} = \boldsymbol{1}$ if edge between node $\boldsymbol{i}$ and $\boldsymbol{j}$




| | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 0 | 1 | 0 |
| 2 | 1 | 0 | 1 | 0 | 0 | 0 |
| 3 | 1 | 1 | 0 | 1 | 0 | 0 |
| 4 | 0 | 0 | 1 | 0 | 1 | 1 |
| 5 | 1 | 0 | 0 | 1 | 0 | 1 |
| 6 | 0 | 0 | 0 | 1 | 1 | 0 |

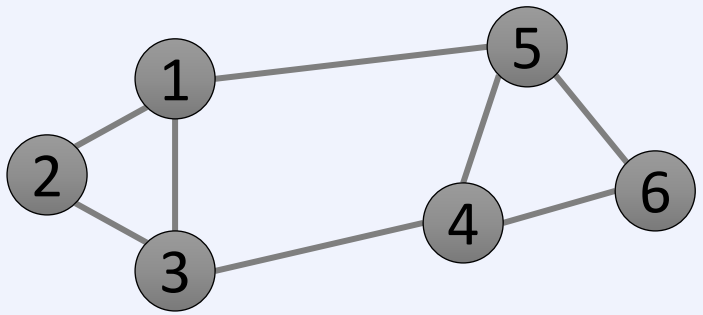
**Important properties:**

- symmetric matrix
- eigenvectors are real and orthogonal

# Matrix representations (2)

The **degree matrix** of a graph is a diagonal $n$-by-$n$ matrix with the degree of vertex $i$ at position $\mathbf{\Delta}_{ii} = d_i$

- $\mathbf{\Delta}_{ii} = d_i = \sum_j a_{ij} =$ degree of node $\boldsymbol{i}$
- $\boldsymbol{n \times n}$ diagonal matrix




| | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 1 | 3 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 2 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 3 | 0 | 0 | 0 |
| 4 | 0 | 0 | 0 | 3 | 0 | 0 |
| 5 | 0 | 0 | 0 | 0 | 3 | 0 |
| 6 | 0 | 0 | 0 | 0 | 0 | 2 |

# Matrix representations (3)

The **normalized adjacency matrix** $\boldsymbol{M}$ is the adjacency matrix where in every row $i$ all values are divided by $d_i$

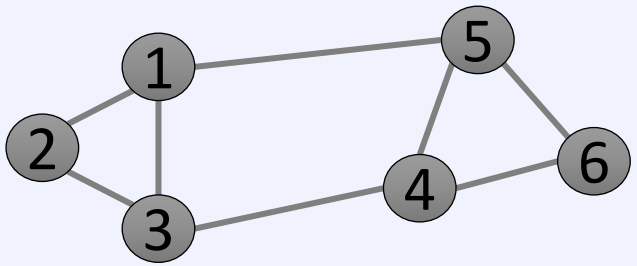
- every row sums up to 1
- $\boldsymbol{M} = \boldsymbol{\Delta}^{-1}\boldsymbol{A}$

**(picture is on vacation)**

# Matrix representations (4)

The **Laplacian matrix** $L$ or $\Lambda$ of a graph is the adjacency matrix subtracted from the degree matrix

- $n \times n$ symmetric matrix




| | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 1 | 3 | -1 | -1 | 0 | -1 | 0 |
| 2 | -1 | 2 | -1 | 0 | 0 | 0 |
| 3 | -1 | -1 | 3 | -1 | 0 | 0 |
| 4 | 0 | 0 | -1 | 3 | -1 | -1 |
| 5 | -1 | 0 | 0 | -1 | 3 | -1 |
| 6 | 0 | 0 | 0 | -1 | -1 | 2 |

$$L = D - A$$

Important properties:

- **eigenvalues** are non-negative real numbers
- **eigenvectors** are real and orthogonal

# Matrix representations (4)

The **Laplacian matrix** $L$ or $\Lambda$ of a graph is the adjacency matrix subtracted from the degree matrix

- $\boldsymbol{n}\times\boldsymbol{n}$ symmetric matrix

$$\boldsymbol{L = \Lambda = \Delta - A} = \begin{pmatrix} \sum_{j\neq 1} a_{1,j} & -a_{1,2} & \cdots & -a_{1,n} \\ -a_{2,1} & \sum_{j\neq 2} a_{2,j} & \cdots & -a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ -a_{n,1} & -a_{n,2} & \cdots & \sum_{j\neq n} a_{n,j} \end{pmatrix}$$

The Laplacian is symmetric and positive semi-definite

- (for undirected graphs)
- has $n$ real, non-negative, orthogonal eigenvalues

$$0 \leq \lambda_1 \leq \lambda_2 \leq \lambda_3 \leq \cdots \lambda_n$$

# The normalised, symmetric Laplacian

The **normalised, symmetric Laplacian matrix** $L^s$ or $\Lambda^s$ of a graph is defined as

$$\boldsymbol{\Delta}^{-\frac{\mathbf{1}}{\mathbf{2}}}\boldsymbol{L}\boldsymbol{\Delta}^{-\frac{\mathbf{1}}{\mathbf{2}}} = \boldsymbol{I} - \boldsymbol{\Delta}^{-\frac{\mathbf{1}}{\mathbf{2}}}\mathbf{A}\boldsymbol{\Delta}^{-\frac{\mathbf{1}}{\mathbf{2}}} = \begin{pmatrix} \frac{\sum_{j\neq 1} a_{1,j}}{\sqrt{d_1 d_1}} & -\frac{a_{1,2}}{\sqrt{d_1 d_2}} & \cdots & -\frac{a_{1,n}}{\sqrt{d_1 d_n}} \\ -\frac{a_{2,1}}{\sqrt{d_2 d_1}} & \frac{\sum_{j\neq 2} a_{2,j}}{\sqrt{d_2 d_2}} & \cdots & -\frac{a_{2,n}}{\sqrt{d_2 d_n}} \\ \vdots & \vdots & \ddots & \vdots \\ -\frac{a_{n,1}}{\sqrt{d_n d_1}} & -\frac{a_{n,2}}{\sqrt{d_n d_2}} & \cdots & \frac{\sum_{j\neq n} a_{n,j}}{\sqrt{d_n d_n}} \end{pmatrix}$$

and is also positive semi-definite

The **normalised, asymmetric Laplacian** $L^a$ is $L^a = \Delta^{-1}L$

# Clusterings and matrices, redux

Recall that we can express a clustering using a binary cluster assignment matrix

- each row has exactly one non-zero

Let the $i$-th column of this matrix be $\boldsymbol{c}_i$

- clusters are disjoint so $\boldsymbol{c}_i^T \boldsymbol{c}_j = 0$
- cluster has $\boldsymbol{c}_i^T \boldsymbol{c}_i = ||\boldsymbol{c}_i||^2$ elements

We can get the $vol(C_i)$ and $W(C_i, V)$ using $\boldsymbol{c_i}$'s

- $vol(C_i) = \sum_{j \in C_i} d_j = \sum_{r=1}^{n} \sum_{s}^{n} \boldsymbol{c}_{ir} \boldsymbol{\Delta}_{rs} \boldsymbol{c}_{is} = \boldsymbol{c}_i^T \boldsymbol{\Delta} \boldsymbol{c}_i$
- $W(C_i, C_i) = \sum_{r \in C_i} \sum_{s \in C_i} a_{rs} = \boldsymbol{c}_i^T \boldsymbol{A} \boldsymbol{c}_i$
- $W(C_i, V \setminus C_i) = W(C_i, V) - W(C_i, C_i) = \boldsymbol{c}_i^T (\boldsymbol{\Delta} - \boldsymbol{A}) \boldsymbol{c}_i = \boldsymbol{c}_i^T \boldsymbol{L} \boldsymbol{c}_i$

# Cuts using matrices

$$\text{RatioCut} = \sum_{i=1}^{k} \frac{W(C_i, V \setminus C_i)}{|C_i|} = \sum_{i=1}^{k} \frac{\boldsymbol{c}_i^T \boldsymbol{L} \boldsymbol{c}_i}{||\boldsymbol{c}_i||^2}$$

$$\text{NormalisedCut} = \sum_{i=1}^{k} \frac{W(C_i, V \setminus C_i)}{vol(C_i)} = \sum_{i=1}^{k} \frac{\boldsymbol{c}_i^T \boldsymbol{L} \boldsymbol{c}_i}{\boldsymbol{c}_i^T \boldsymbol{\Delta} \boldsymbol{c}_i}$$

# The second eigenvalue, $\lambda_2$, as an optimization problem

Fact: for symmetric matrix $\boldsymbol{M}$:

$$\lambda_2 = \min_{\boldsymbol{x}} \frac{\boldsymbol{x}^T \boldsymbol{M} \boldsymbol{x}}{\boldsymbol{x}^T \boldsymbol{x}}$$

What is the meaning of $\min \boldsymbol{x}^T \boldsymbol{L} \boldsymbol{x}$ on graph $G$?

$$\boldsymbol{x}^T \boldsymbol{L} \boldsymbol{x} = \sum_{ij}^{n} \boldsymbol{L}_{ij} \boldsymbol{x}_i \boldsymbol{x}_j = \sum_{ij}^{n} (\boldsymbol{D}_{ij} - \boldsymbol{A}_{ij}) \boldsymbol{x}_i \boldsymbol{x}_j$$

$$= \sum_{i} \boldsymbol{D}_{ii} \boldsymbol{x}_i^2 - \sum_{(i,j) \in E} 2 \boldsymbol{x}_i \boldsymbol{x}_j$$

$$= \sum_{(i,j) \in E} (\boldsymbol{x}_i^2 + \boldsymbol{x}_j^2 - 2 \boldsymbol{x}_i \boldsymbol{x}_j) \qquad = \sum_{(i,j) \in E} (\boldsymbol{x}_i - \boldsymbol{x}_j)^2$$

Node $i$ has degree $d_i$. So, value $\boldsymbol{x}_i^2$ needs to be summed up $d_i$ times.
But each edge $(i, j)$ has two endpoints, so we need $\boldsymbol{x}_i^2 + \boldsymbol{x}_j^2$

# $\lambda_2$, as an optimization problem

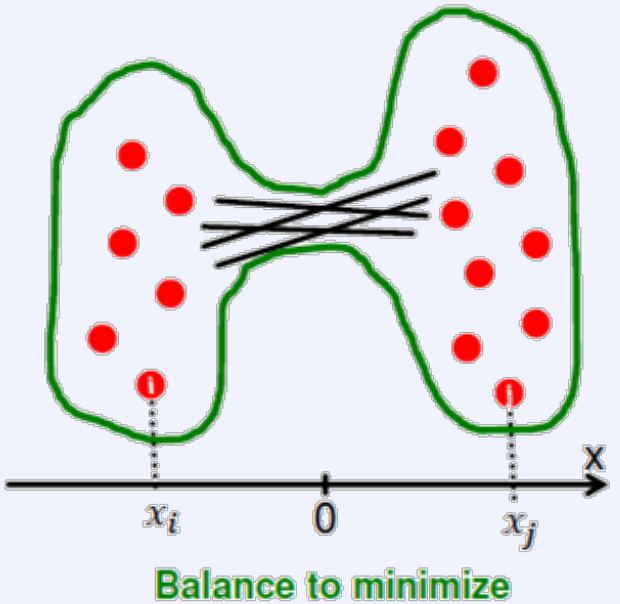
What else do we know about $x$?

- $\boldsymbol{x}$ is a unit vector: $\sum_i \boldsymbol{x}_i^2 = 1$
- $\boldsymbol{x}$ is orthogonal to 1st eigenvector $(1, \ldots, 1)$ thus: $\sum_i x_i \cdot 1 = \sum_i x_i = 0$

Remember

$$\lambda_2 = \min_{\substack{\text{all labelings} \\ \text{of nodes i so} \\ \text{that } \sum \boldsymbol{x}_i = 0}} \frac{\sum_{(i,j)\in E} \left(\boldsymbol{x}_i - \boldsymbol{x}_j\right)^2}{\sum_i \boldsymbol{x}_i^2}$$

We want to assign values $\boldsymbol{x}_i$ to nodes $i$ such that few edges cross 0.
(we want $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ to subtract eachother)



# Finding an optimal cut

Back to finding the optimal cut

Express partition $(C_1, C_2)$ as a vector

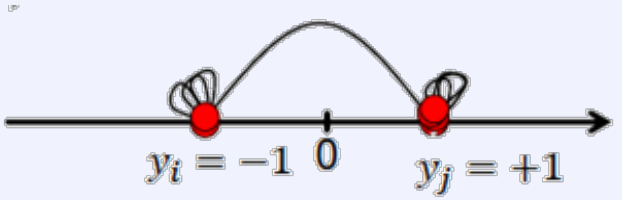
$$\boldsymbol{c}_i = \begin{cases} +1 & \text{if } i \in C_1 \\ -1 & \text{if } i \in C_2 \end{cases}$$

We can **minimise the cut of the partition** by finding a non-trivial vector $x$ that **minimises**

$$\arg \min_{\boldsymbol{c} \in [-1,+1]^n} f(\boldsymbol{c}) = \sum_{(i,j) \in E} (\boldsymbol{c}_i - \boldsymbol{c}_j)^2$$

NP-hard… so, let's relax!

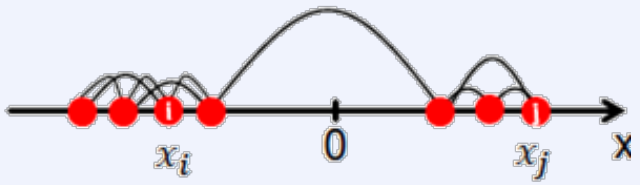
- let $\boldsymbol{c}_i$'s take any real value




(Fiedler, 1973)

# Rayleigh theorem

$$\min_{\boldsymbol{c}\in\mathbb{R}^n} f(\boldsymbol{c}) = \sum_{(i,j)\in E} \left(\boldsymbol{c}_i - \boldsymbol{c}_j\right)^2 = \boldsymbol{c}^T \boldsymbol{L}\boldsymbol{c}$$




$\lambda_2 = \min_{\boldsymbol{c}} f(\boldsymbol{c})$: the minimum value of $f(\boldsymbol{c})$ is given by the 2nd smallest eigenvalue $\lambda_2$ of the Laplacian matrix $\boldsymbol{L}$

$\boldsymbol{x} = \arg\min_{\boldsymbol{c}} f(\boldsymbol{c})$: the optimal solution for $\boldsymbol{c}$ is given by the corresponding eigenvector $\boldsymbol{x}$ and is referred to as the Fiedler vector

# So far…

How to define a **good** partition of a graph?

- minimise a given graph cut criterion

How to efficiently identify such a partition?

- approximate using information provided by the eigenvalues and eigenvalues of a graph

Spectral clustering

# Spectral clustering algorithms

Three basic stages

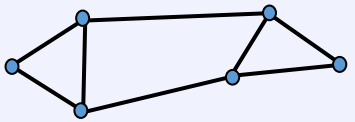
1. pre-processing
   - construct a matrix representation of the graph
2. decomposition
   - compute eigenvalues and eigenvectors of the matrix
   - map each point to a lower-dimensional representation based on one or more eigenvectors
3. grouping
   - assign points to two or more clusters, based on the new representation

# Spectral partitioning algorithm

1) **Pre-processing:**
- build Laplacian matrix $\boldsymbol{L}$ of the graph

2) **Decomposition:**
- find eigenvalues $\lambda$ and eigenvectors $\boldsymbol{x}$ of the matrix $\boldsymbol{L}$

- map vertices to corresponding components of $\lambda_2$



| | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 1 | 3 | -1 | -1 | 0 | -1 | 0 |
| 2 | -1 | 2 | -1 | 0 | 0 | 0 |
| 3 | -1 | -1 | 3 | -1 | 0 | 0 |
| 4 | 0 | 0 | -1 | 3 | -1 | -1 |
| 5 | -1 | 0 | 0 | -1 | 3 | -1 |
| 6 | 0 | 0 | 0 | -1 | -1 | 2 |

$\lambda =$

| |
|---|
| 0.0 |
| 1.0 |
| 3.0 |
| 3.0 |
| 4.0 |
| 5.0 |

$\mathbf{X} =$

| | | | | | |
|---|---|---|---|---|---|
| 0.4 | 0.3 | -0.5 | -0.2 | -0.4 | -0.5 |
| 0.4 | 0.6 | 0.4 | -0.4 | 0.4 | 0.0 |
| 0.4 | 0.3 | 0.1 | 0.6 | -0.4 | 0.5 |
| 0.4 | -0.3 | 0.1 | 0.6 | 0.4 | -0.5 |
| 0.4 | -0.3 | -0.5 | -0.2 | 0.4 | 0.5 |
| 0.4 | -0.6 | 0.4 | -0.4 | -0.4 | 0.0 |

| | |
|---|---|
| **1** | 0.3 |
| **2** | 0.6 |
| **3** | 0.3 |
| **4** | -0.3 |
| **5** | -0.3 |
| **6** | -0.6 |

How do we now find the clusters?

# Spectral partitioning

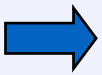
## 3) Grouping:

- sort components of reduced 1-dimensional vector
- identify clusters by splitting the sorted vector in two

## How to choose a splitting point?

- naïve approaches:
  - split at **0** or median value
- more expensive approaches:
  - attempt to minimize normalized cut in 1-dimension (sweep over ordering of nodes induced by the eigenvector)

| | |
|---|---|
| 1 | 0.3 |
| 2 | 0.6 |
| 3 | 0.3 |
| 4 | -0.3 |
| 5 | -0.3 |
| 6 | -0.6 |



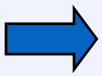
Split at 0:

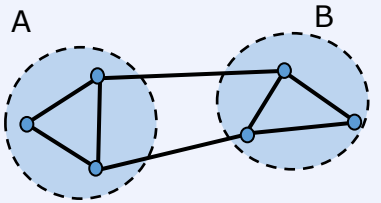
**Cluster A:** Positive points

**Cluster B:** Negative points

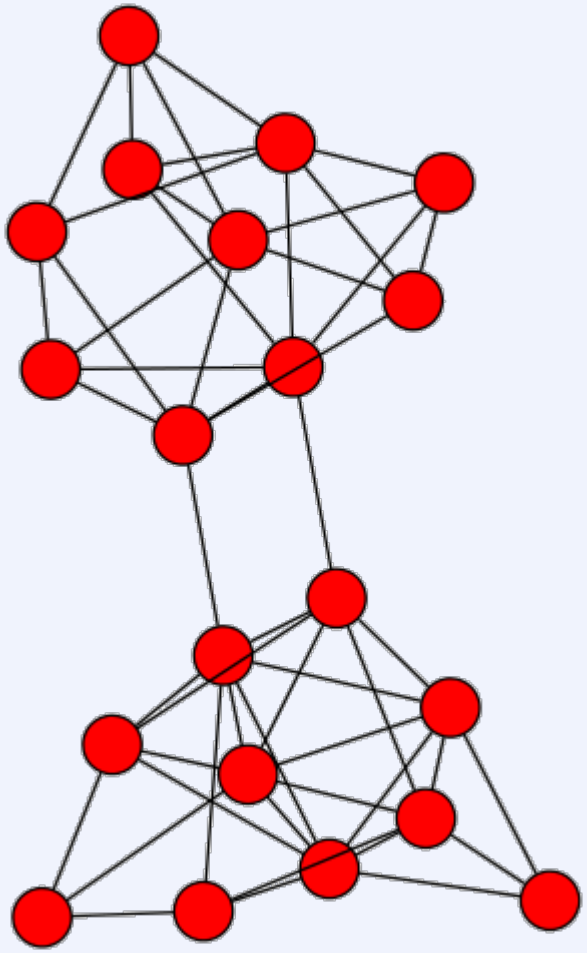
| | |
|---|---|
| 1 | 0.3 |
| 2 | 0.6 |
| 3 | 0.3 |

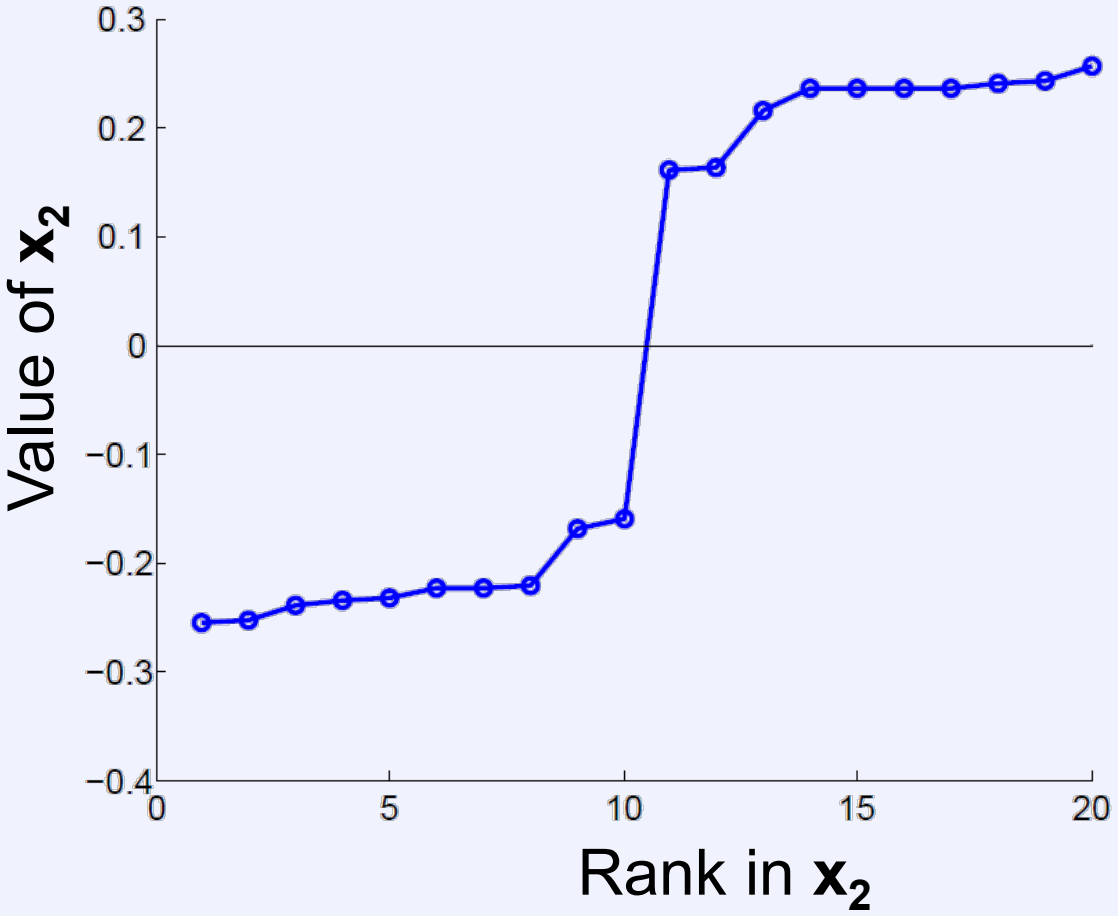
| | |
|---|---|
| 4 | -0.3 |
| 5 | -0.3 |
| 6 | -0.6 |



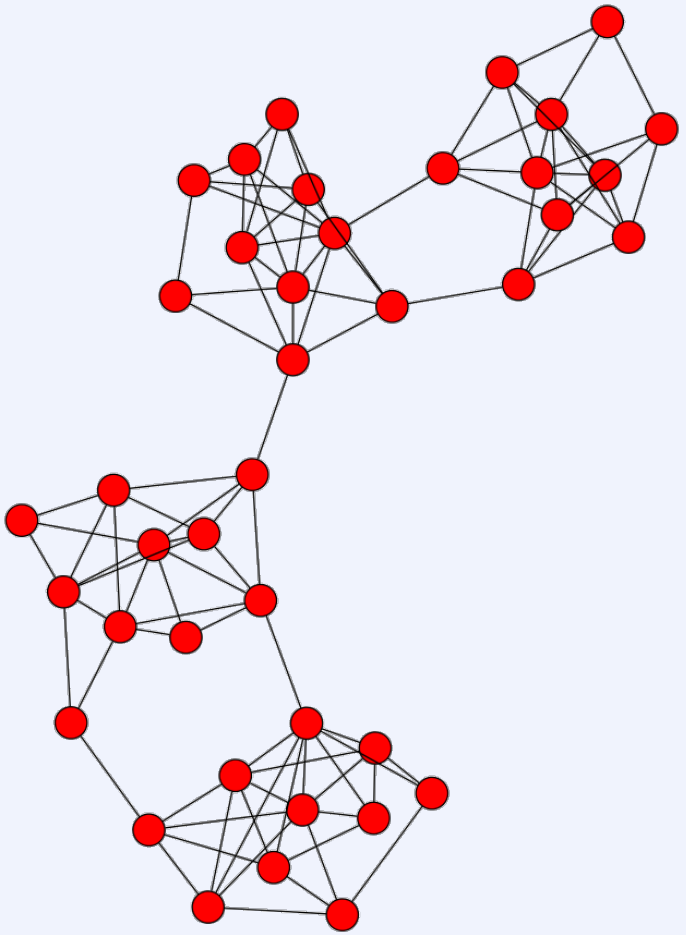


# Example: Spectral Partitioning

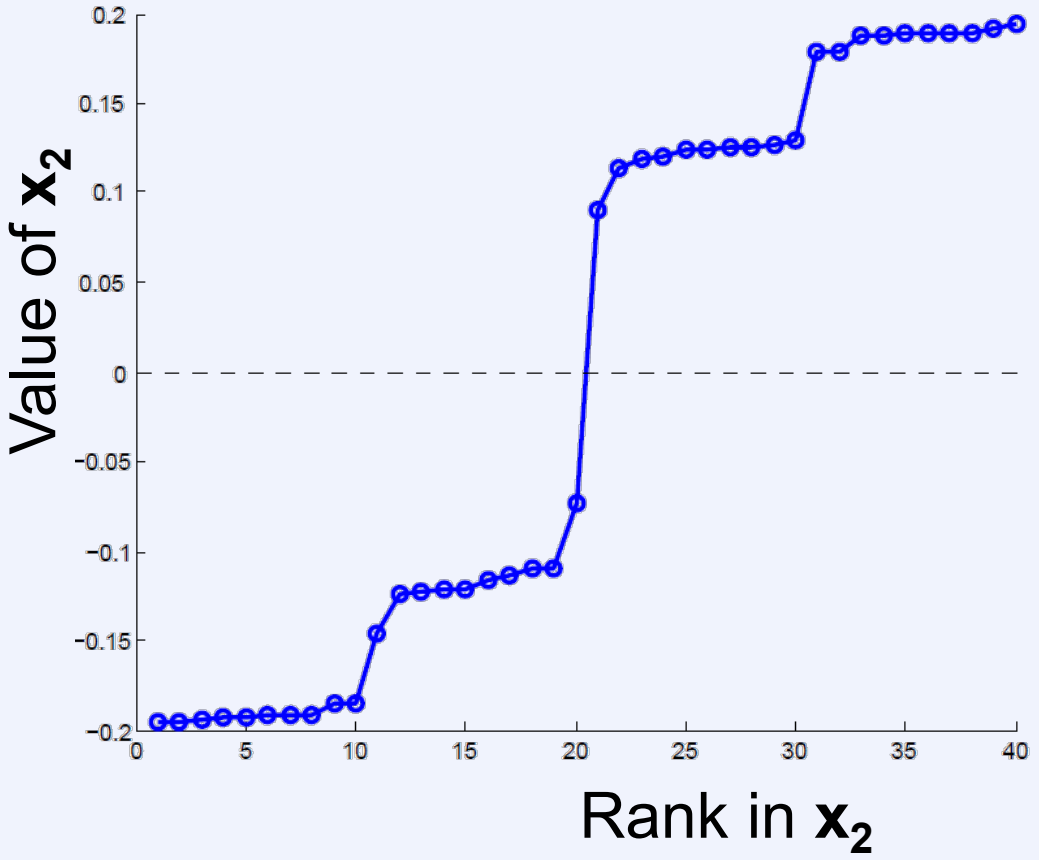




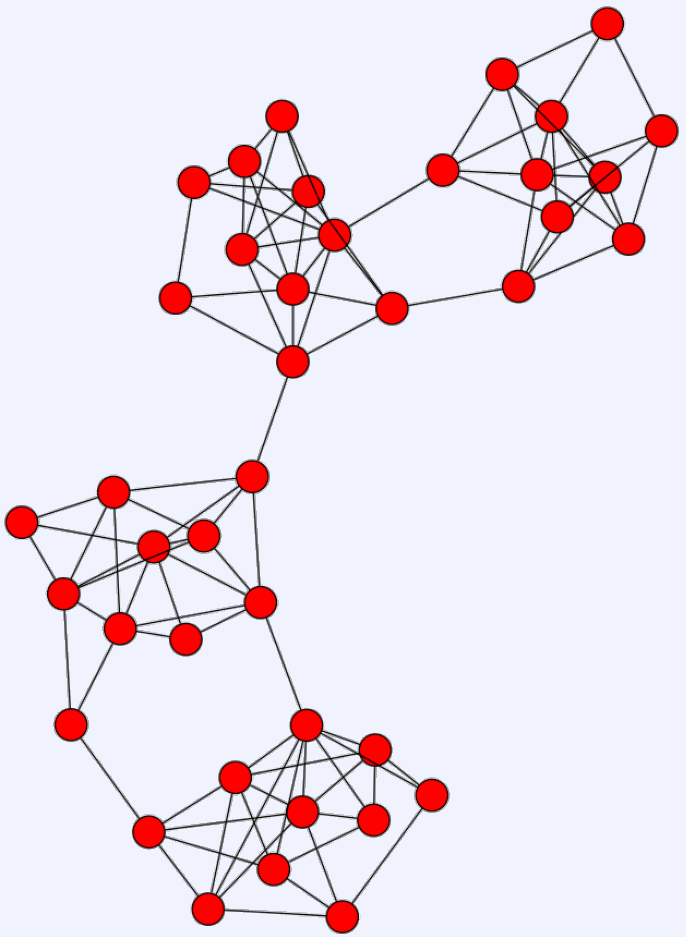
# Example: Spectral Partitioning



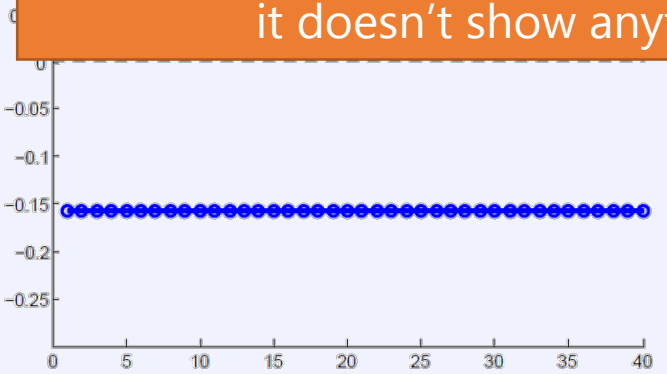
Eigenvector corresponding to $\lambda_2$ is **useful**, it shows communities!

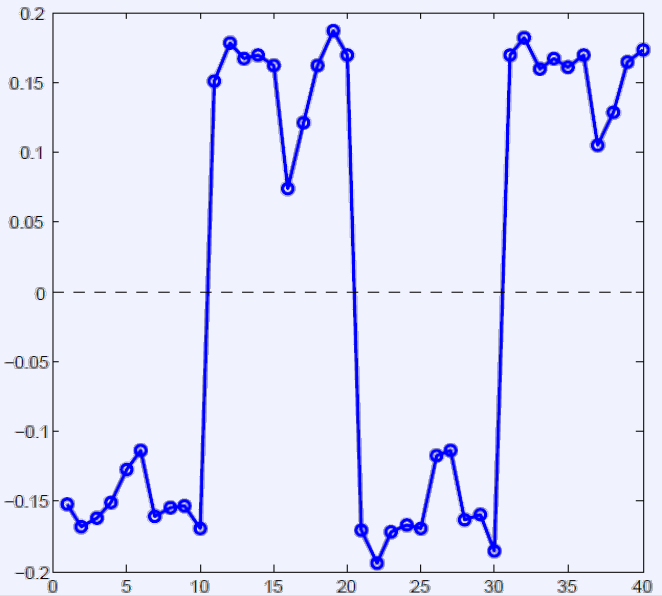


# Example: Spectral Partitioning



Eigenvector corresponding to $\lambda_1$ is **useless**, it doesn't show anything




Eigenvector corresponding to $\lambda_3$ is **useless** by itself, but **useful** when considered **together with $\lambda_2$**

# $k$-way Spectral clustering

How do we partition a graph into $k$ clusters?

There are two basic approaches

- **recursive bi-partitioning** (Hagen et al., '92)
  - recursively apply a bi-partitioning algorithm in a hierarchical divisive manner
  - inefficient, and unstable

- **cluster multiple eigenvectors** (Shi-Malik, '00)
  - build a reduced space from multiple eigenvectors
  - commonly used in recent papers
  - a preferable approach…

# Why use multiple eigenvectors?

## Approximates the optimal cut

- can be used to approximate optimal $k$-way normalized cut

## Emphasizes cohesive clusters

- increases the unevenness in the distribution of the data
- associations between similar points are amplified, associations between dissimilar points are attenuated
- the data begins to "approximate a clustering"

## Well-separated space

- transforms data to a new "embedded space", consisting of $\boldsymbol{k}$ orthogonal basis vectors

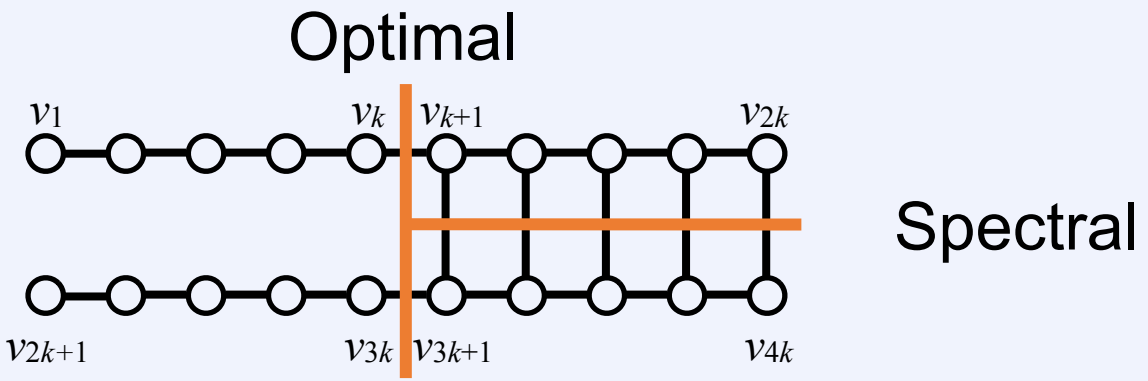
## Multiple eigenvectors prevent instability due to information loss

(Shi-Malik, 2000)

# Is spectral clustering optimal?

Spectral clustering is not always a good approximation of the graph cuts

- in so-called cockroach graphs, spectral clustering always cuts horizontally, while vertically is optimal
- approximation ratio of $O(n)$



# Spectral clustering

To do the clustering, we need to move our real-valued eigenvectors $\boldsymbol{u}_i$ to binary cluster indicator vectors

First, create a matrix $\boldsymbol{U}$ with $\boldsymbol{u}_i$'s as its columns

- optionally, normalize the rows to sum up to 1 (esp when using $\boldsymbol{L}^s$)

Cluster the rows of this matrix using $k$-means

- or, in principle, any other clustering algorithm

Solving the eigenvectors is $O(n^3)$ in general or $O(n^2)$ if the similarity graph has as many edges as vertices

- the $k$-means on the $\boldsymbol{U}$ matrix takes $O(tnk^2)$
  - $t$ is the number of iterations in $k$-means

# Another look at Approximate cut

Allowing for real-valued cluster assignment vectors $c_i$ makes Relaxed RatioCut look like

$$J_{rc}(C) = \sum_{i=1}^{k} \frac{\boldsymbol{c}_i^T \boldsymbol{L} \boldsymbol{c}_i}{||\boldsymbol{c}_i||^2} = \sum_{i=1}^{k} \left(\frac{\boldsymbol{c}_i}{||\boldsymbol{c}_i||}\right)^T \boldsymbol{L} \left(\frac{\boldsymbol{c}_i}{||\boldsymbol{c}_i||}\right) = \sum_{i=1}^{k} \boldsymbol{u}_i^T \boldsymbol{L} \boldsymbol{u}_i$$

- $\boldsymbol{u}_i = \frac{\boldsymbol{c}_i}{||\boldsymbol{c}_i||}$ i.e. the unit vector in the direction of $\boldsymbol{c}_i$

# Solving the relaxed version

We want to minimize the function $J_{rc}$ over $\boldsymbol{u}_i$'s

- we have a constraint that $\boldsymbol{u}_i^T \boldsymbol{u}_i = 1$

To solve, derive w.r.t. $\boldsymbol{u}_i$'s and find the roots

- add Lagrange multipliers to incorporate the constraints:

$$\frac{\delta}{\delta \boldsymbol{u}_i}\left(\sum_i^k \boldsymbol{u}_i^T \boldsymbol{L}\boldsymbol{u}_i + \sum_i^k \lambda_i\left(1 - \boldsymbol{u}_i^T \boldsymbol{u}_i\right)\right) = \boldsymbol{0}$$

Hence, $\boldsymbol{L}\boldsymbol{u}_i = \lambda_i \boldsymbol{u}_i$

- $\boldsymbol{u}_i$ is an eigenvector of $\boldsymbol{L}$ corresponding to the eigenvalue $\lambda_i$

# Which eigenvectors to choose

We know that $\boldsymbol{L}\boldsymbol{u}_i = \lambda_i \boldsymbol{u}_i$

- hence $\lambda_i = \boldsymbol{u}_i^T \boldsymbol{L}\boldsymbol{u}_i$

As we're minimizing the sum of $\boldsymbol{u}_i^T \boldsymbol{L}\boldsymbol{u}_i$'s we should choose the $\boldsymbol{u}_i$'s corresponding to the $k$ smallest eigenvalues

- these are our **relaxed cluster indicators**

But, we also know that $\lambda_1 = 0$ and
that the corresponding eigenvector is $(n^{-\frac{1}{2}}, n^{-\frac{1}{2}}, \ldots, n^{-\frac{1}{2}})$

- hmm, that doesn't help with clustering...

# Normalised cut and choice of Lap

For normalized cut similar procedure shows that we should select the $k$ smallest eigenvectors of $\boldsymbol{L}^s$ instead of $\boldsymbol{L}$

- or, we can use the asymmetric Laplacian $\boldsymbol{L}^a$

Which one we should choose?

- both ratio and normalised cut aim at minimising intra-cluster similarity
- only normalised cut considers inter-cluster similarity → either $\boldsymbol{L}^s$ or $\boldsymbol{L}^a$

The asymmetric Laplacian is better

- with symmetric one further normalisation is needed

# Pseudo-code

**Algorithm** SPECTRALCLUSTERING(connected graph $G, k$) :

compute the similarity matrix $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ for $G$

**if** $ratio\ cut$ **then** $\boldsymbol{B} \leftarrow \boldsymbol{L}$

**else if** $normalised\ cut$ **then** $\boldsymbol{B} \leftarrow \boldsymbol{L^s}$ or $\boldsymbol{L^a}$

solve $\boldsymbol{B u}_i = \lambda_i \boldsymbol{u}_i$ for $i = k + 1$, where $\lambda_2 \leq \lambda_3 \leq \cdots \leq \lambda_{k+1}$

$\boldsymbol{U} \leftarrow (\boldsymbol{u}_n \ \ \boldsymbol{u}_{n-1} \ \ \ldots \boldsymbol{u}_{n-k+1})$

$\boldsymbol{Y} \leftarrow$ normalise rows of $\boldsymbol{U}$

$\boldsymbol{C} \leftarrow \{\boldsymbol{C_1}, \ldots, \boldsymbol{C_k}\}$ via $k$-means on $\boldsymbol{Y}$

# Conclusions

Frequent subgraph mining finds recurring patterns in graph data

- enormously complex problem → exact algorithms can't be fast
- but graphs are not usually very big even if there are many of them

Graph clustering is much like other clustering

- any clusterable data can be turned into similarity graph
- spectral clustering uses well-known linear algebra
- though this doesn't necessarily make it a good clustering algorithm

# *Thank you!*

Frequent subgraph mining finds recurring patterns in graph data

- enormously complex problem → exact algorithms can't be fast
- but graphs are not usually very big even if there are many of them

Graph clustering is much like other clustering

- any clusterable data can be turned into similarity graph
- spectral clustering uses well-known linear algebra
- though this doesn't necessarily make it a good clustering algorithm