Online Social Networks and Media

Graph Partitioning:

cuts, spectral clustering, density

What we have seen so far (in PART I)

1. Introduction: what, why, types?
2. Cliques (clique percolation method)
3. Background: How it relates to "cluster analysis" (node/edge similarity) (partitioned (k-means) and hierarchical clustering, how to embed nodes)
4. Betweeness centrality (divisive algorithm)
5. Modularity (intracommunity edges vs random), label propagation

Outline

PART II

Cuts

Spectral Clustering

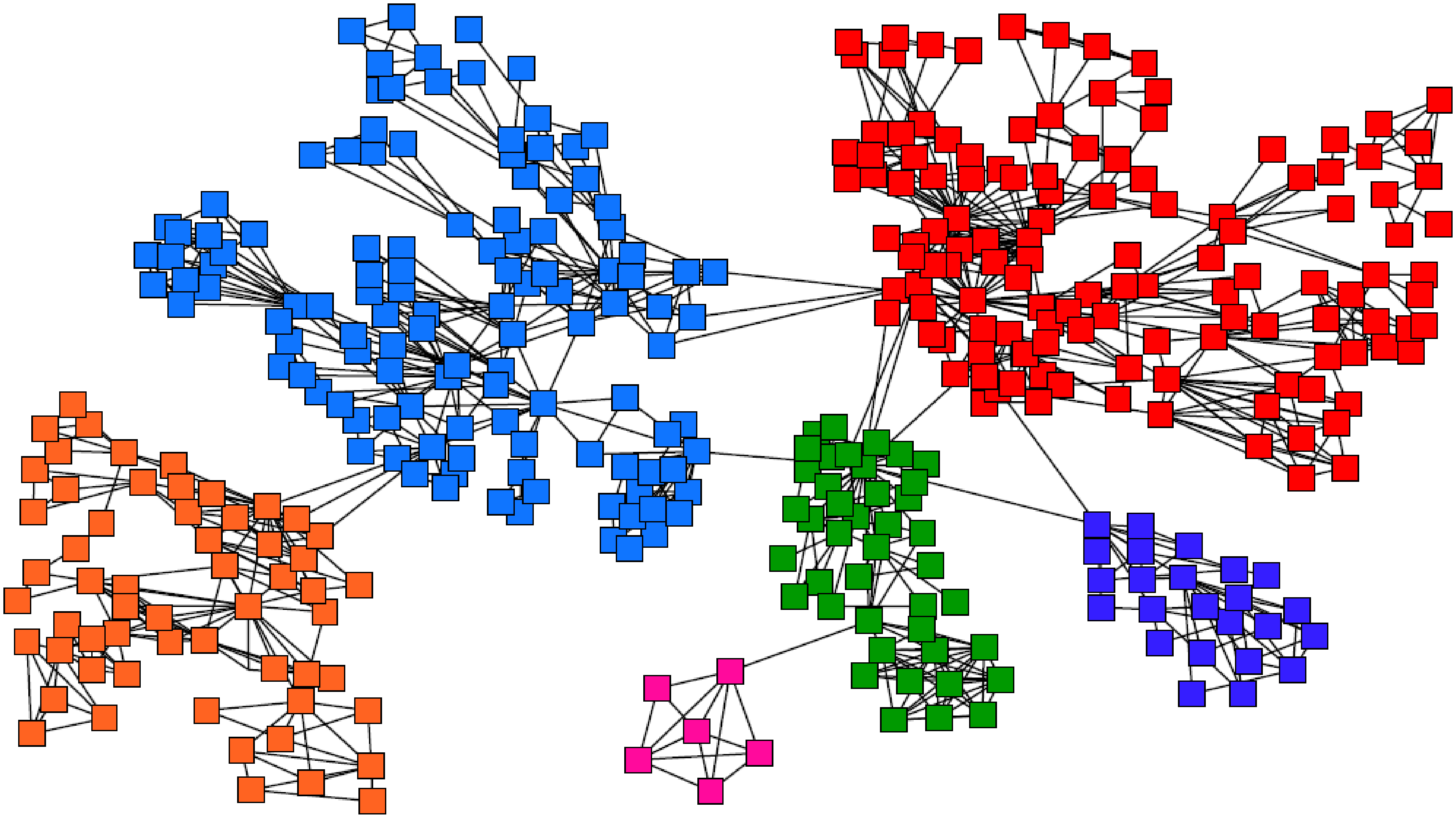
Dense Subgraphs

Evaluation

We cut the graph into several partitions and assume these partitions represent communities

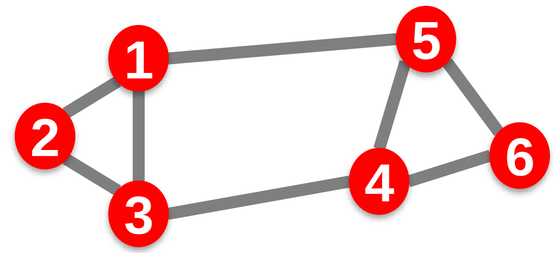
- **Cut**: partitioning (*cut*) of the graph into two (or more) sets (*cutsets*)
  - The **size of the cut** is the **number of edges** that are being cut

Graph Partitioning



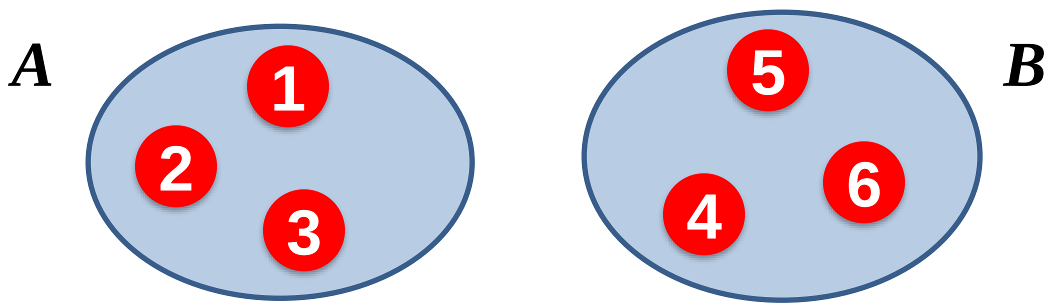
Graph Partitioning

Undirected graph $G(V, E)$:



Bi-partitioning task:

Divide vertices into two disjoint groups $\boldsymbol{A}, \boldsymbol{B}$

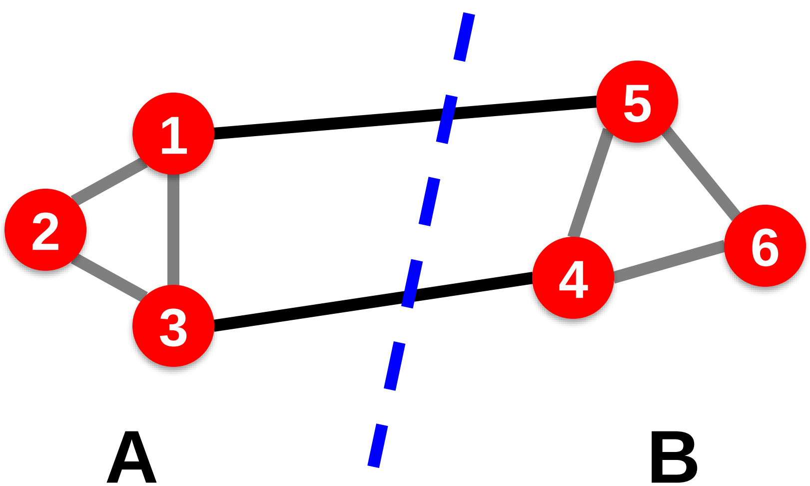


How can we define a "good" partition of $\boldsymbol{G}$*?*

Graph Partitioning

What makes a good partition?

- Maximize the number of within-group connections
- Minimize the number of between-group connections

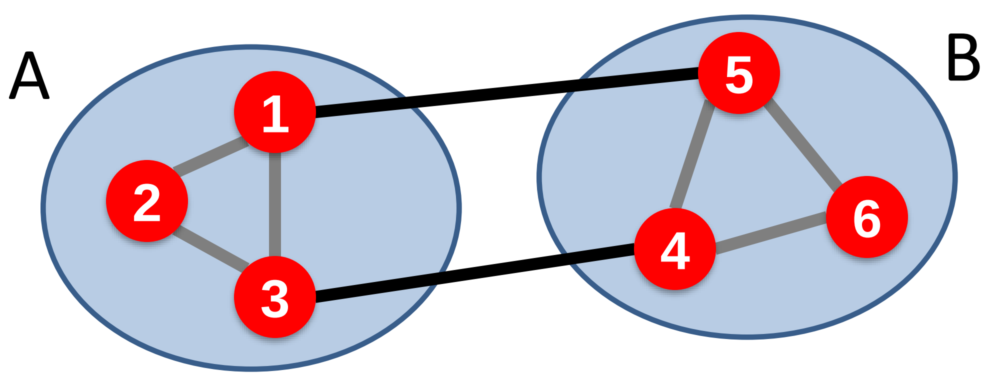


Graph Cuts

Express *partitioning objectives* as a function of the "edge cut" of the partition

Cut: Set of edges with only one vertex in a group:

$$cut(A,B) = \sum_{i \in A, j \in B} w_{ij}$$



cut(A,B) = 2

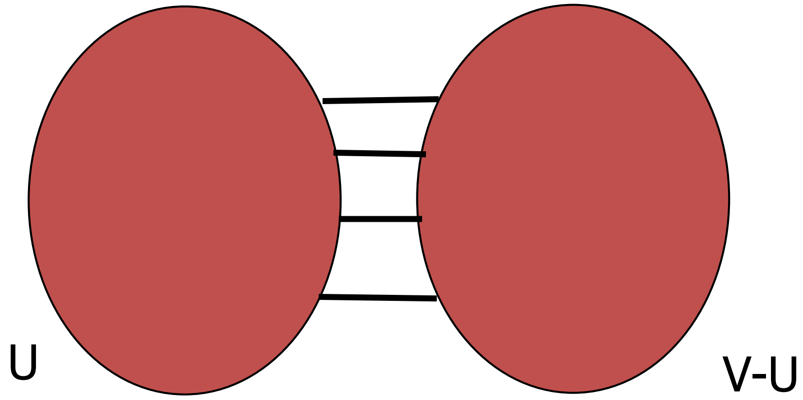
Min Cut

min-cut: the *min number of edges* such that when removed cause the graph to become *disconnected*

Minimizes the number of connections between partition

$$\arg\min_{A,\,B} \mathrm{cut}(A, B)$$

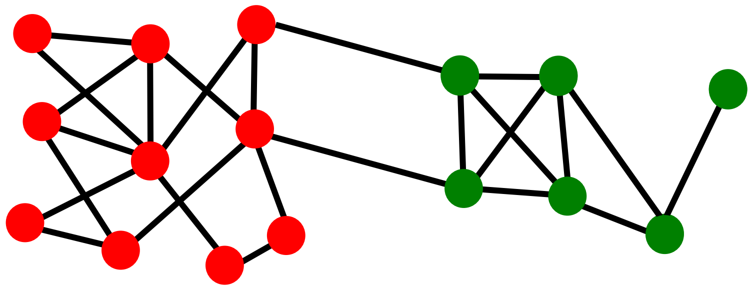
$$\min_{U} E(U, V-U) = \sum_{i \in U} \sum_{j \in V-U} A[i, j]$$



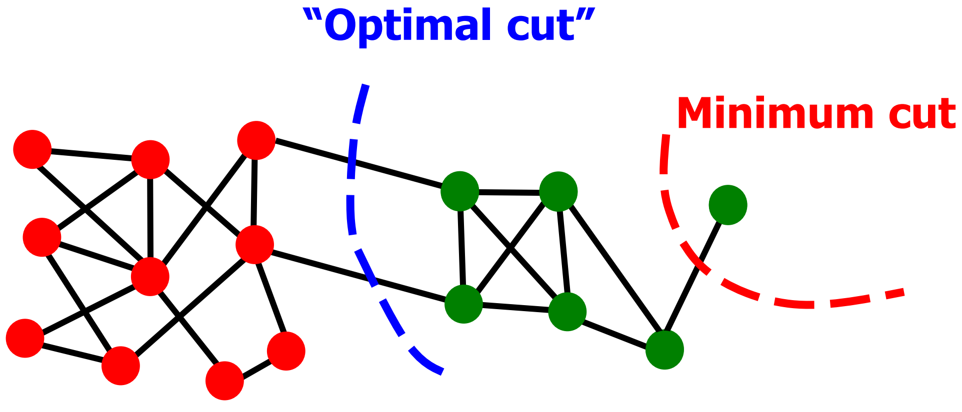
This problem can be solved in polynomial time

Min-cut/Max-flow algorithm

Does this work?



Min Cut



Problem:

- Only considers external cluster connections
- Does not consider *internal* cluster connectivity

Graph Bisection

- Since the minimum cut does not always yield good results, we need *extra constraints* to make the problem meaningful.
- Graph Bisection refers to the problem of partitioning the nodes of the graph into two *equal sets*.

Ratio Cut

Ratio Cut

Normalize cut by the *size* of the groups

$$\text{RatioCut} = \frac{\text{Cut(U,V}-\text{U)}}{|U|} + \frac{\text{Cut(U,V}-\text{U)}}{|V-U|}$$

Normalized Cut

Normalized-cut

Connectivity between groups relative to the *density* of each group

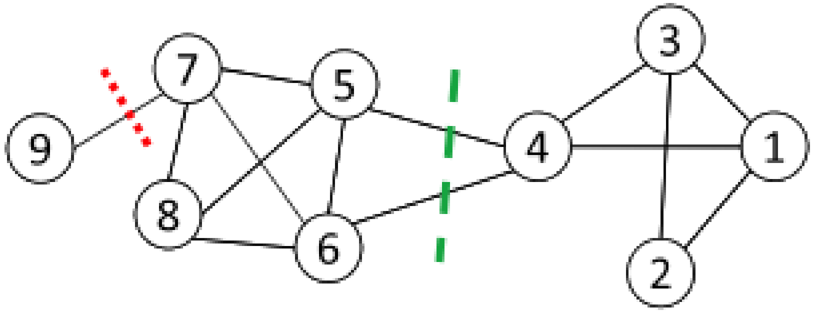
$$\text{Normalized-cut} = \frac{\text{Cut(U,V}-\text{U)}}{Vol(U)} + \frac{\text{Cut(U,V}-\text{U)}}{Vol(V-U)}$$

$vol(U)$: total weight of the edges with at least one endpoint in U, $vol(U) = \sum_{i \in U} d_i$

Why use these criteria?

- Produce more balanced partitions

An example



Min-Cut(Red) = 1

Ratio-Cut(Red) = $\frac{1}{1} + \frac{1}{8} = \frac{9}{8} = 1.125$

Normalized-Cut(Red) = $\frac{1}{1} + \frac{1}{27} = \frac{28}{27} = 1.04$

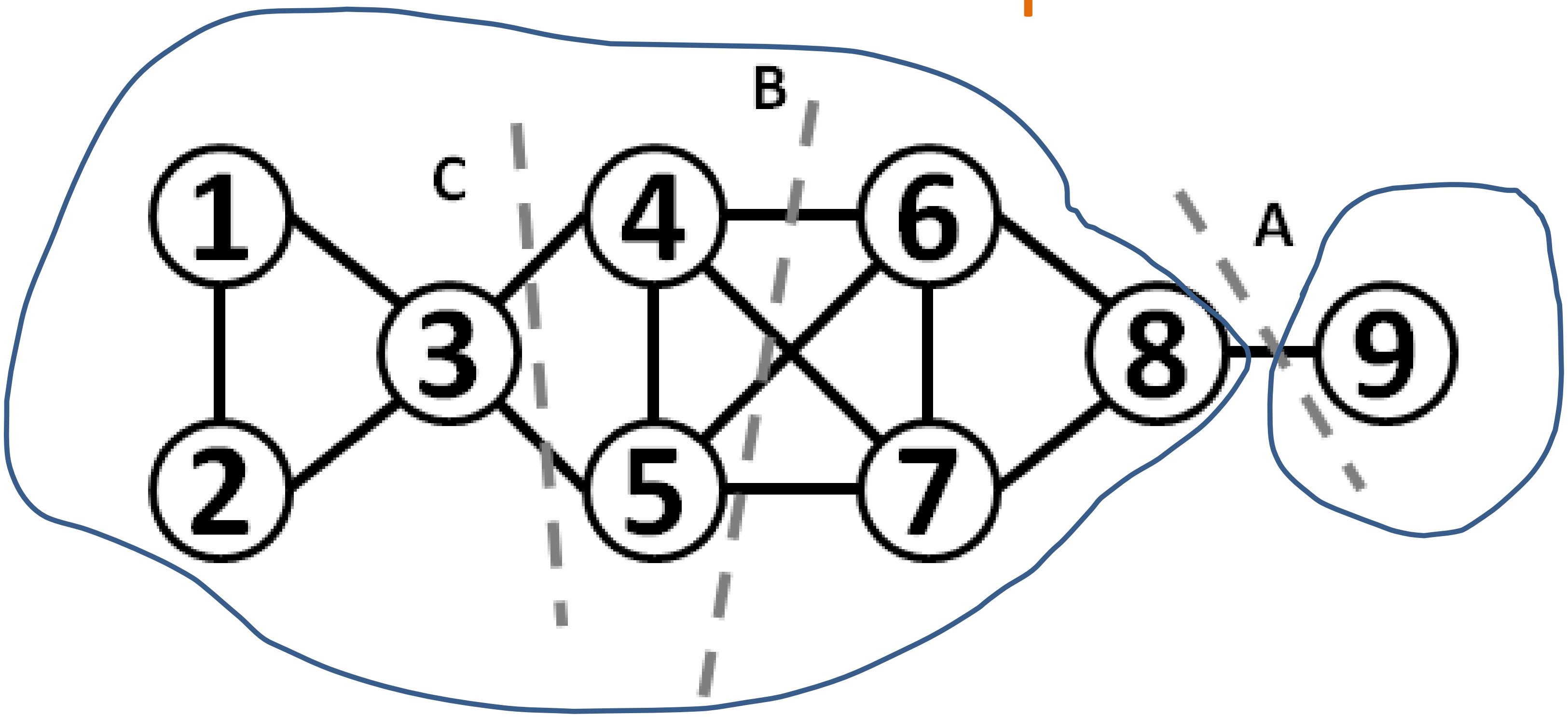
Min-Cut(Green) = 2

Ratio-Cut(Green) = $\frac{2}{5} + \frac{2}{4} = \frac{18}{20} = 0.9$

Normalized-Cut(Green) = $\frac{2}{12} + \frac{2}{16} = \frac{14}{48} = 0.29$

Normalized is smaller due to density

An example

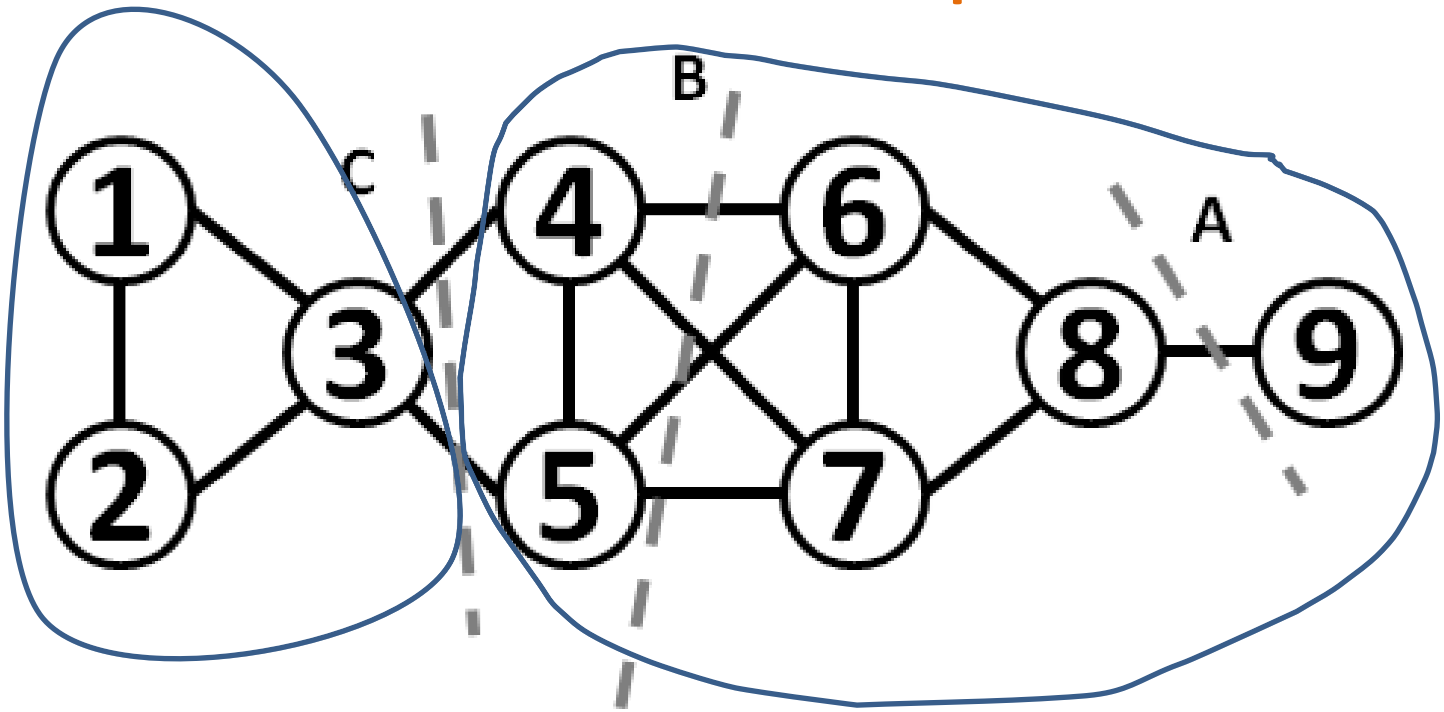


Min-Cut(A) = 1

Min-Cut(B) = 4

Min-Rut(C) = 2

An example

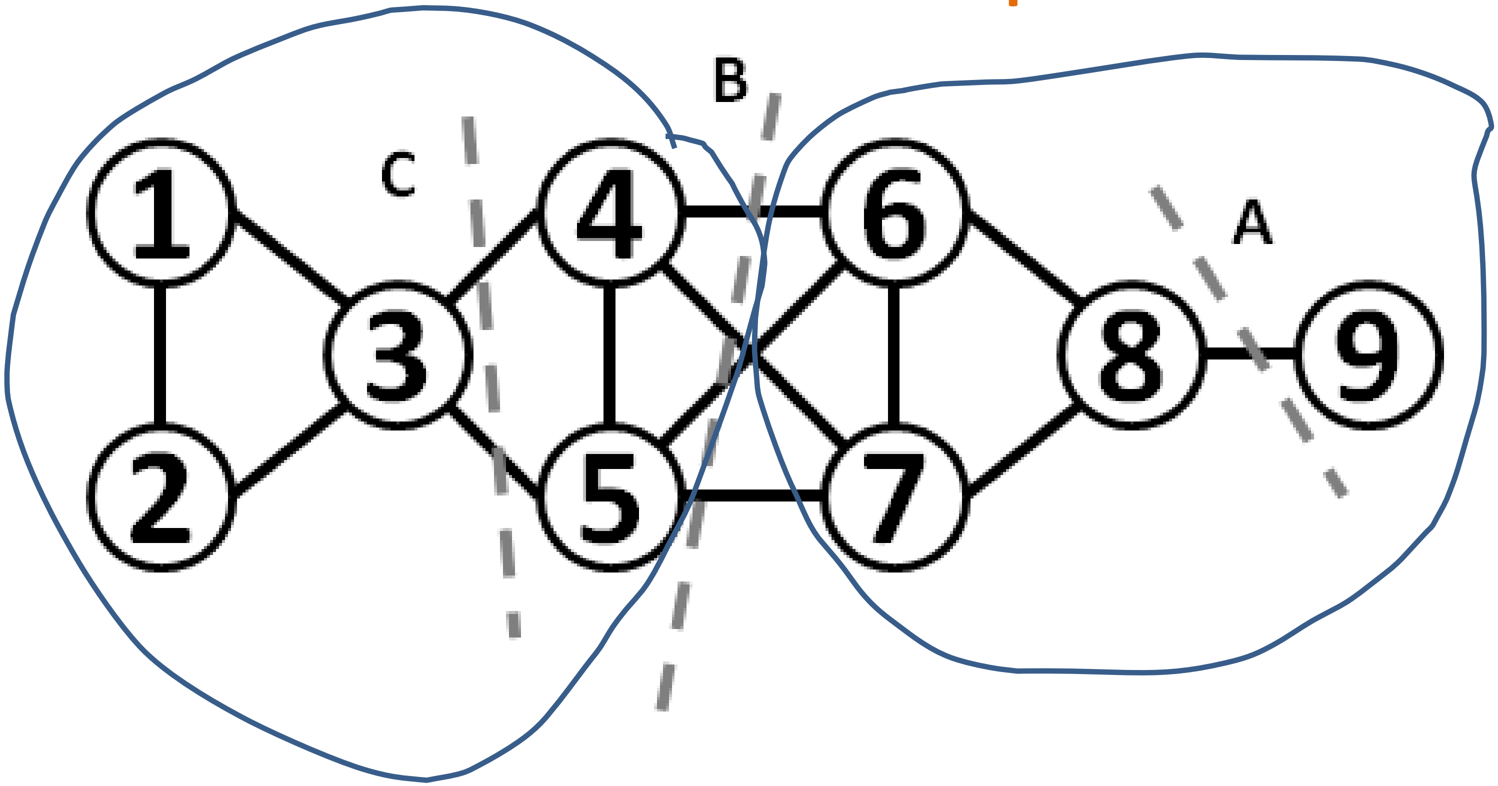


Ratio-Cut(A) = $\frac{1}{1} + \frac{1}{8} = \frac{9}{8} = 1.125$

Ratio-Cut(B) = $\frac{4}{5} + \frac{4}{4} = \frac{36}{20} = 1.8$

Ratio-Rut(C) = $\frac{2}{3} + \frac{2}{6} = \frac{6}{6} = 1$

An example



Normalized-Cut(A) = $\frac{1}{1} + \frac{1}{27} = \frac{28}{27} = 1.04$

Normalized-Cut(B) = $\frac{4}{16} + \frac{4}{12} = \frac{7}{12} = 0.58$

Normalized-Rut(C) = $\frac{2}{8} + \frac{2}{20} = \frac{44}{40} = 1.1$

Graph conductance

Connectivity of group A with the rest of the network relative to the density of the group

$$\varphi(A) = \frac{\text{cut}(A, V - A)}{\min\{\text{vol}(A), 2m - \text{vol}(A)\}}$$

The lower the conductance, the better the cluster

Graph Bisection

The problem find a partition with **equal number** of nodes and **minimum cut** is NP-hard

- Kernighan-Lin algorithm: Start with random equal partitions and then swap nodes to improve some quality metric (e.g., cut, modularity, etc).

Graph Cuts

Ratio and normalized cuts can be reformulated in matrix format and solved using spectral clustering

Outline

PART II

Cuts

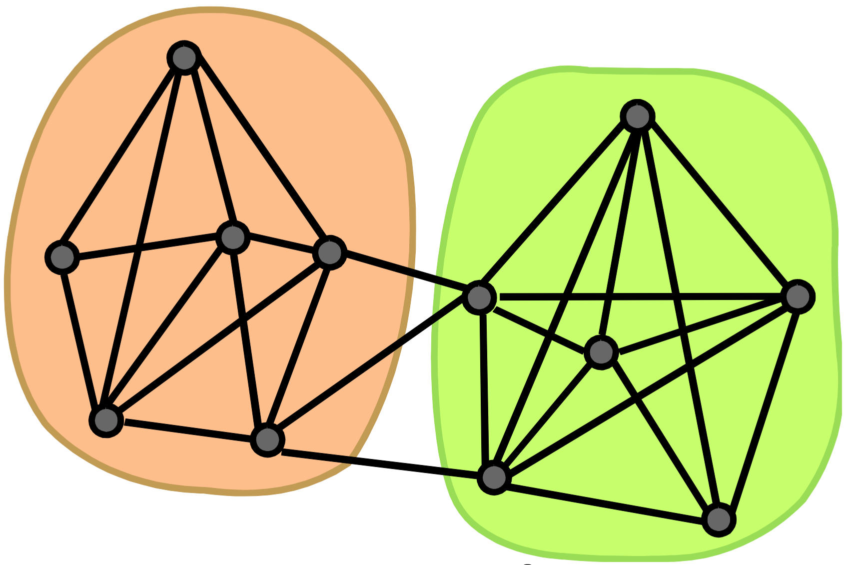
Spectral Clustering

Dense Subgraphs

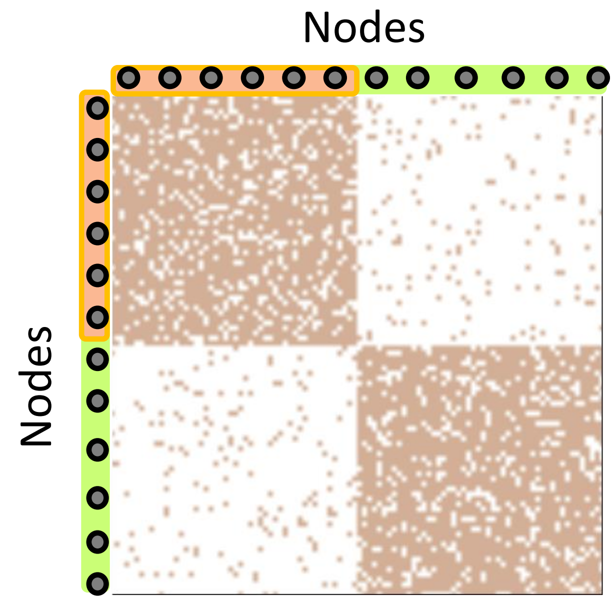
Evaluation

Adjacency matrix

Simplest form: Split the graph into two pieces, many connections within, few across



Network



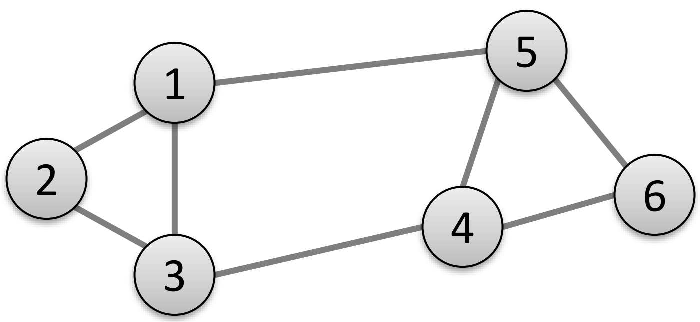
Adjacency matrix

How do we **identify this structure**?
Partition the graph, so that the resulting pieces have **low conductance**

Matrix Representation

Adjacency matrix (A):

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

How many non-zeros in each row?

If the graph is weighted, $a_{ij}= w_{ij}$

Spectral Graph Partitioning

$\boldsymbol{x}$ is a vector in $\Re^n$ with components $(\boldsymbol{x_1}, \ldots, \boldsymbol{x_n})$

- Think of it as a label/value of each node of $\boldsymbol{G}$
  - Value x_i corresponds to node i in the graph

- What is the meaning of $A \cdot x$?

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

$$y_i = \sum_{j=1}^{n} A_{ij} x_j = \sum_{(i,j) \in E} x_j$$

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

Spectral Analysis

i^{th} coordinate of $A \cdot x$:

- Sum of the x-values of neighbors of i
- Make this a new value at node j

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

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

Spectral Graph Theory:

- Analyze the "spectrum" of a matrix representing G
- Spectrum: Eigenvectors x_i of a graph, ordered by the magnitude (strength) of their corresponding eigenvalues λ_i: $\Lambda = \{\lambda_1, \lambda_2, \ldots, \lambda_n\}$ $\lambda_1 \le \lambda_2 \le \ldots \le \lambda_n$

Spectral clustering: use the eigenvectors of *A* or *graphs derived* by it

Most based on the graph Laplacian

Example: d-regular graph

Suppose all nodes in G have degree d and G is connected

- What are some eigenvalues/vectors of G?

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

  – Let's try: $x = (1,1, \dots, 1)$

  – Then: $A \cdot x = (d, d, \dots, d) = \lambda \cdot x$. So: $\lambda = d$

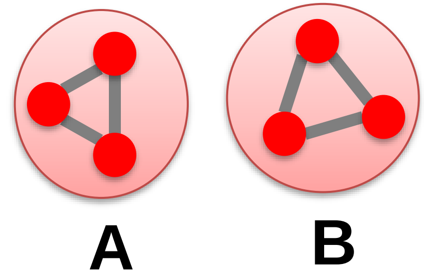
  – We found eigenpair of G: $x = (1,1, \dots, 1), \lambda = d$

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

$$y_j = \sum_{i=1}^{n} A_{ij} x_i = \sum_{(j,i) \in E} x_i$$

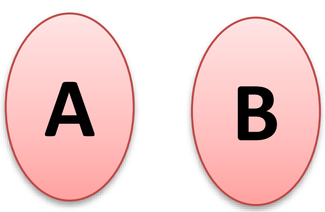
Example: Graph on 2 components

- What if G is not connected?
  - G has 2 components, each d-regular

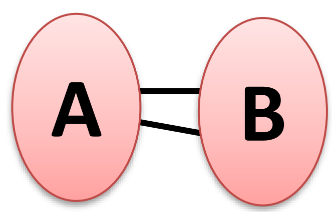


- What are some eigenvectors?
  - $x =$ Put all $\mathbf{1}$s on $\boldsymbol{A}$ and $\mathbf{0}$s on $\boldsymbol{B}$ or vice versa
    - $x' = (\underbrace{1,\dots,1}_{|A|},\underbrace{0,\dots,0}_{|B|})$ then $\mathrm{A} \cdot x' = (d,\dots,d,0,\dots,0)$
    - $x'' = (0,\dots,0,1,\dots,1)$ then $A \cdot x'' = (0,\dots,0,d,\dots,d)$
    - And so in both cases the corresponding $\lambda = d$

- A bit of intuition:



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



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

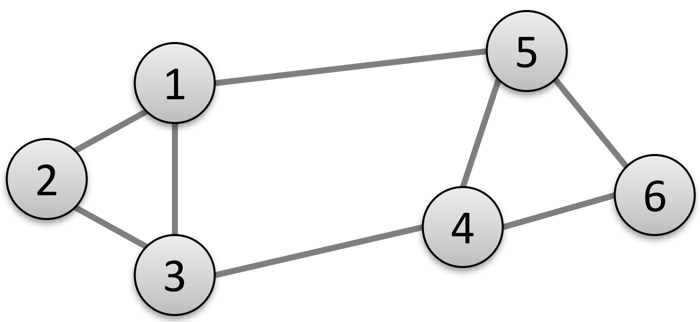
2nd largest eigenvalue λ_{n-1} now has value very close to λ_n

What is the right matrix to apply this intuition?

Matrix Representations

Adjacency matrix (A):

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

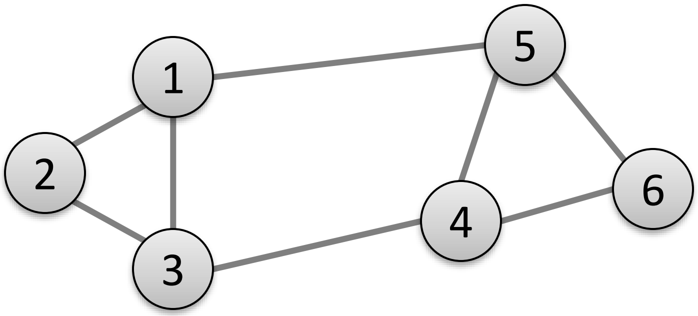
Important properties:

- Symmetric matrix
- Eigenvectors are real and orthogonal

Matrix Representations

Degree matrix (D):

- $n \times n$ diagonal matrix
- $D=[d_{ii}]$, d_{ii} = degree of node i



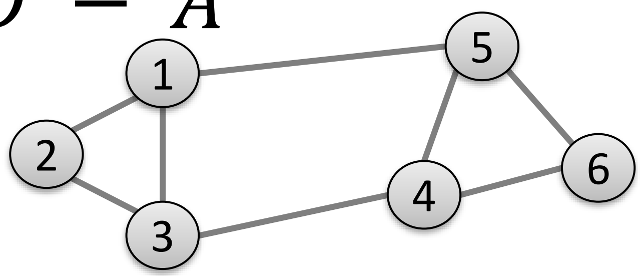
	1	2	3	4	5	6
1	3	0	0	0	0	0
2	0	2	0	0	0	0
3	0	0	3	0	0	0
4	0	0	0	3	0	0
5	0	0	0	0	3	0
6	0	0	0	0	0	2

Graph Laplacian

Laplacian matrix (L):

– $n \times n$ symmetric matrix

$$L = D - A$$



	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

- What is trivial eigenpair?
  - $\boldsymbol{x} = (\mathbf{1}, \ldots, \mathbf{1})$ then $\boldsymbol{L} \cdot \boldsymbol{x} = \mathbf{0}$ and so $\boldsymbol{\lambda} = \boldsymbol{\lambda}_{\mathbf{1}} = \mathbf{0}$
- Important properties:
  - Eigenvalues are non-negative real numbers
  - Eigenvectors are real and orthogonal

Graph Laplacian

If the graph is disconnected

- If there are two connected components, the same argument as for the adjacency matrix applies, and $\lambda_1 = \lambda_2 = 0$
- The multiplicity of eigenvalue 0 is equal to the number of connected components

The second smallest eigenvalue

Fact: For a symmetric matrix M

$$\lambda_2 = \min_x \frac{x^T M\ x}{x^T x}$$

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

λ_2 as an optimization problem

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

- $\mathrm{x}^{\mathrm{T}}\mathrm{L}\,\mathrm{x} = \sum_{i,j=1}^{n} L_{ij}\, x_i x_j = \sum_{i,j=1}^{n} \left(D_{ij} - A_{ij}\right) x_i x_j$
- $= \sum_i D_{ii} x_i^2 - \sum_{(i,j)\in E} 2 x_i x_j$
- $= \sum_{(i,j)\in E} (x_i^2 + x_j^2 - 2 x_i x_j) = \sum_{(i,j)\in E} \left(x_i - x_j\right)^2$

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

λ_2 as an optimization problem

The expression: $x^T L x$

is

$$\sum_{(i,j)\in E} \left(x_i - x_j\right)^2$$

When is this expression minimized?
“similar values” for connected edges

λ_2 as an optimization problem

What else do we know about *x*?

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

$$\lambda_2 = \min \frac{\sum_{(i,j)\in E} (x_i - x_j)^2}{\sum_i x_i^2} \quad =1$$

All labelings of nodes i so that $\sum x_i = 0$

If i and j are connected, we want x_i and x_j to subtract each other, have the “same sign”

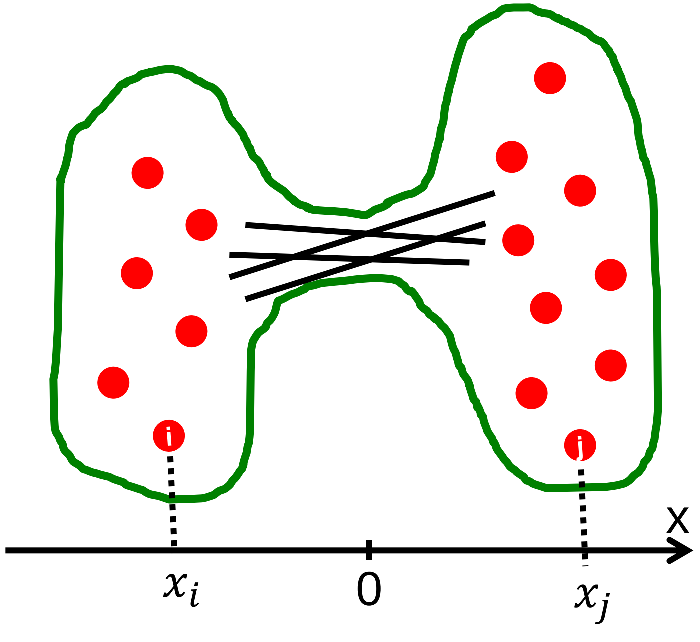
We want to assign values x_i to nodes *i* such that few edges cross 0.

λ_2 as an optimization problem

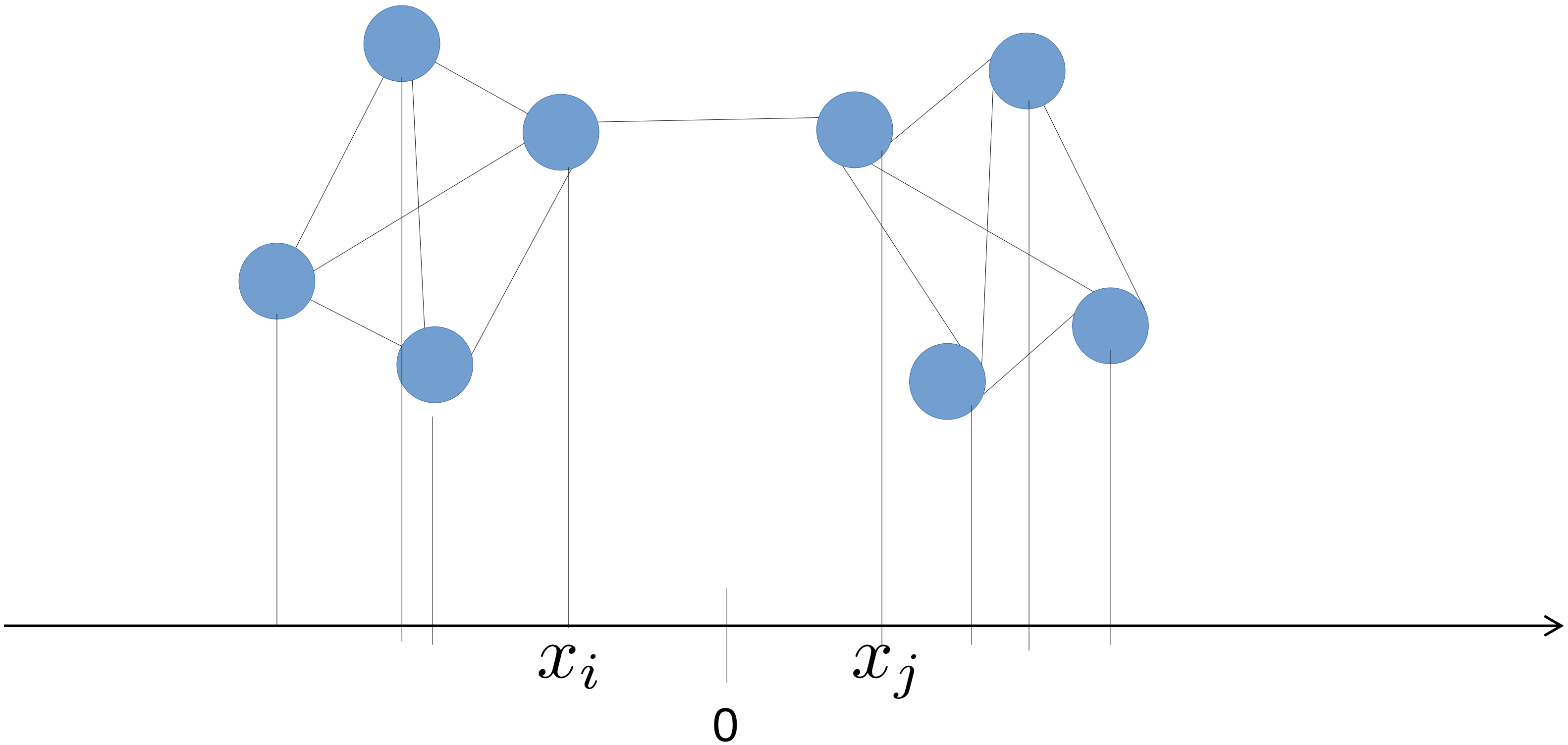
$$\lambda_2 = \min \frac{\sum_{(i,j)\in E} (x_i - x_j)^2}{\sum_i x_i^2}$$

All labelings
of nodes i so
that $\sum x_i = 0$

- Minimum when connected nodes get the same sign (similar values)
- This minimization problem tries to place (embed) nodes of the graph on the real line so that the number of edges that span across 0 is as small as possible
- Tightly connected nodes on the same side of the real line



$$\lambda_2 = \min_{x: \sum x_i = 0} \sum_{(i,j) \in E} (x_i - x_j)^2$$



Find Optimal Cut [Fiedler'73]

Back to finding the optimal cut

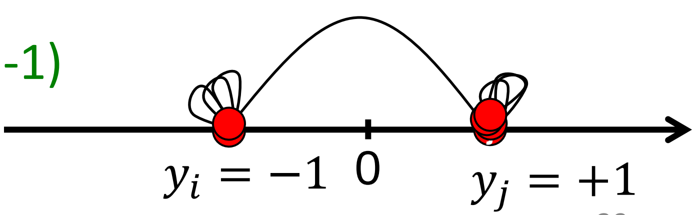
- Express partition (A,B) as a vector

$$y_i = \begin{cases} +1 & if\ i \in A \\ -1 & if\ i \in B \end{cases}$$

- We can minimize the cut of the partition by finding a non-trivial vector x that minimizes:

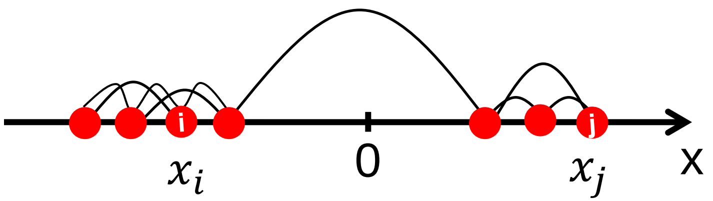
$$\arg\min_{y \in [-1,+1]^n} f(y) = \sum_{(i,j) \in E} (y_i - y_j)^2$$

Can't solve exactly. Let's relax y and allow it to take any real value (instead of just +1, -1)



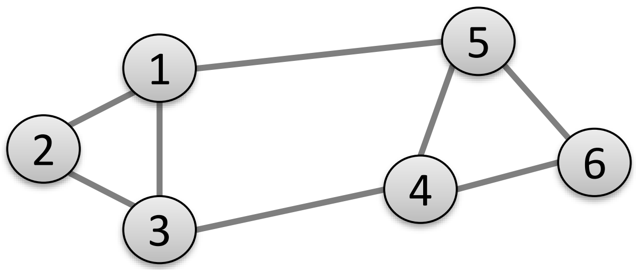
Rayleigh Theorem

$$\min_{y \in \Re^n} f(y) = \sum_{(i,j)\in E} (y_i - y_j)^2 = y^T L y$$



- $\lambda_2 = \min_y f(y)$: The minimum value of $f(y)$ is given by the 2nd smallest eigenvalue λ_2 of the Laplacian matrix L
- $x = \arg\min_y f(y)$: The optimal solution for y is given by the corresponding eigenvector x, referred as the Fiedler vector

Example



Eigenvalues

0.0	1.0	3.0	3..0	4..0	5.0

Eigenvectors

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

Spectral Partitioning Algorithm

Three basic stages:

Pre-processing

- Construct a matrix representation of the graph

Decomposition

- Compute eigenvalues and eigenvectors of the matrix

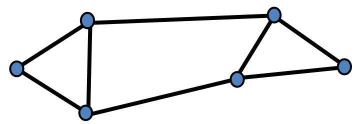
Grouping

- Assign points to two or more clusters, based on the new representation

Spectral Partitioning Algorithm

Pre-processing:

Build Laplacian matrix L of the graph



	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

Decomposition:

– Find eigenvalues λ and eigenvectors x of the matrix L

λ =

0.0
1.0
3.0
3.0
4.0
5.0

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

– Map vertices to corresponding components of λ_2

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 Algorithm

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

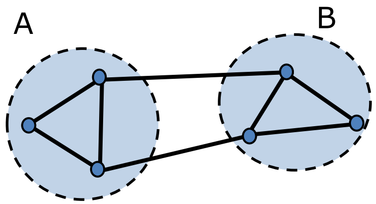
Split at 0:

Cluster A: Positive points

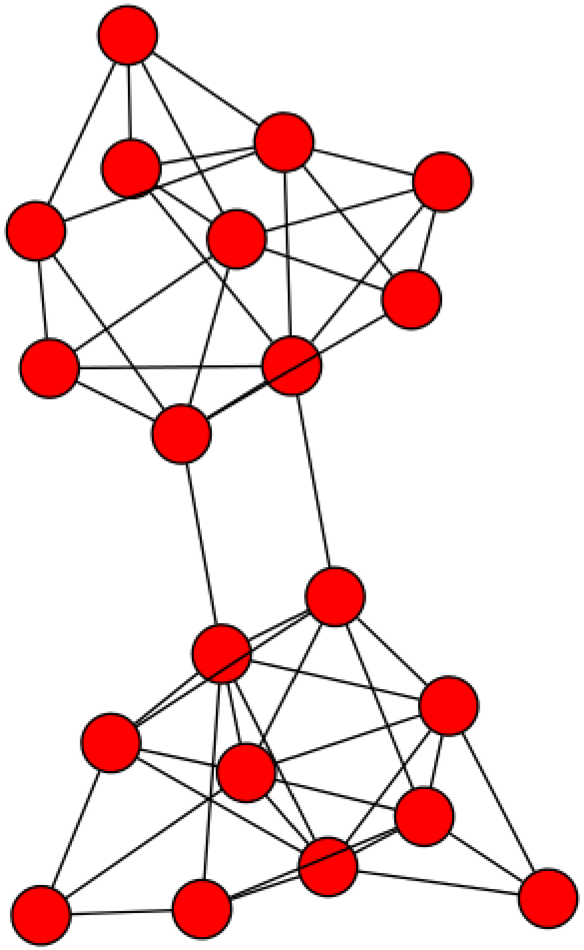
Cluster B: Negative points

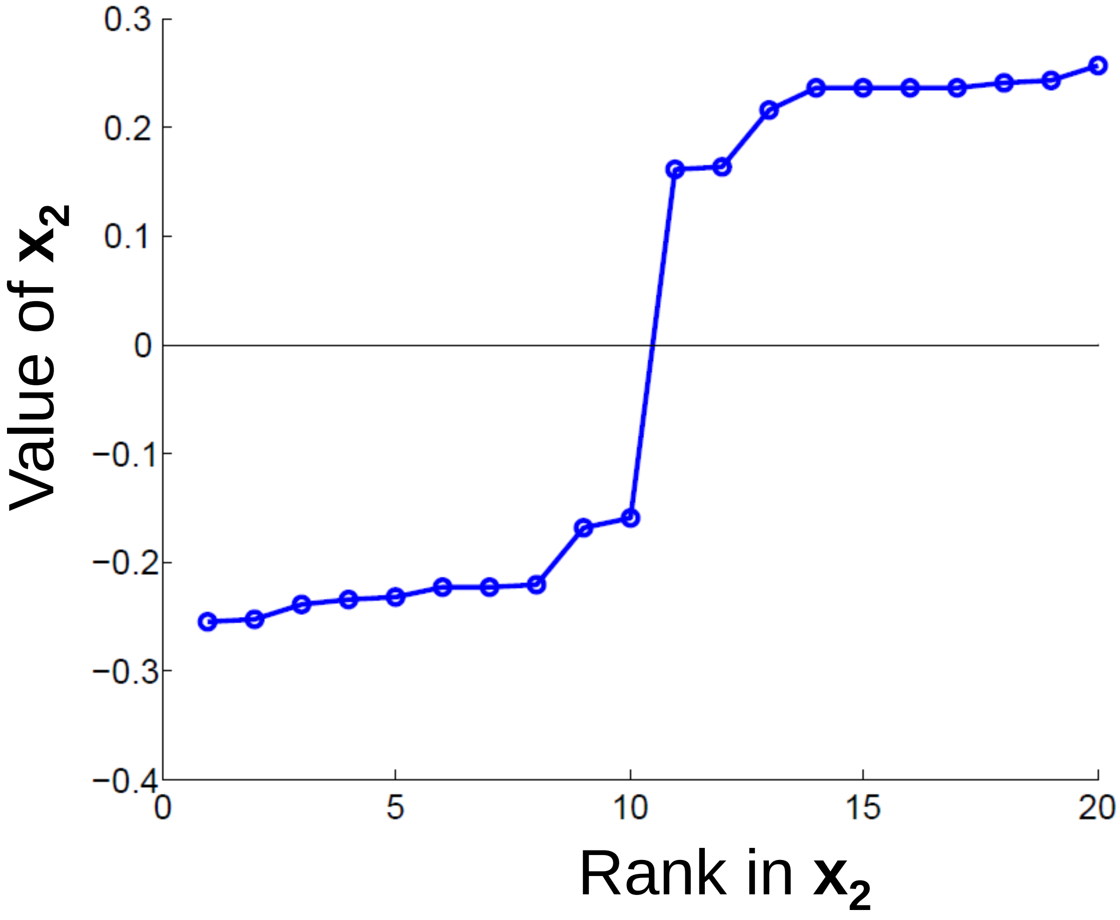
1	0.3
2	0.6
3	0.3

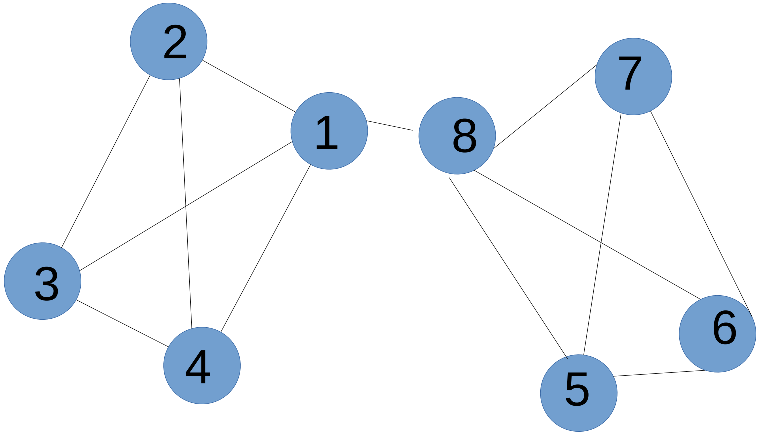
4	-0.3
5	-0.3
6	-0.6



Example: Spectral Partitioning



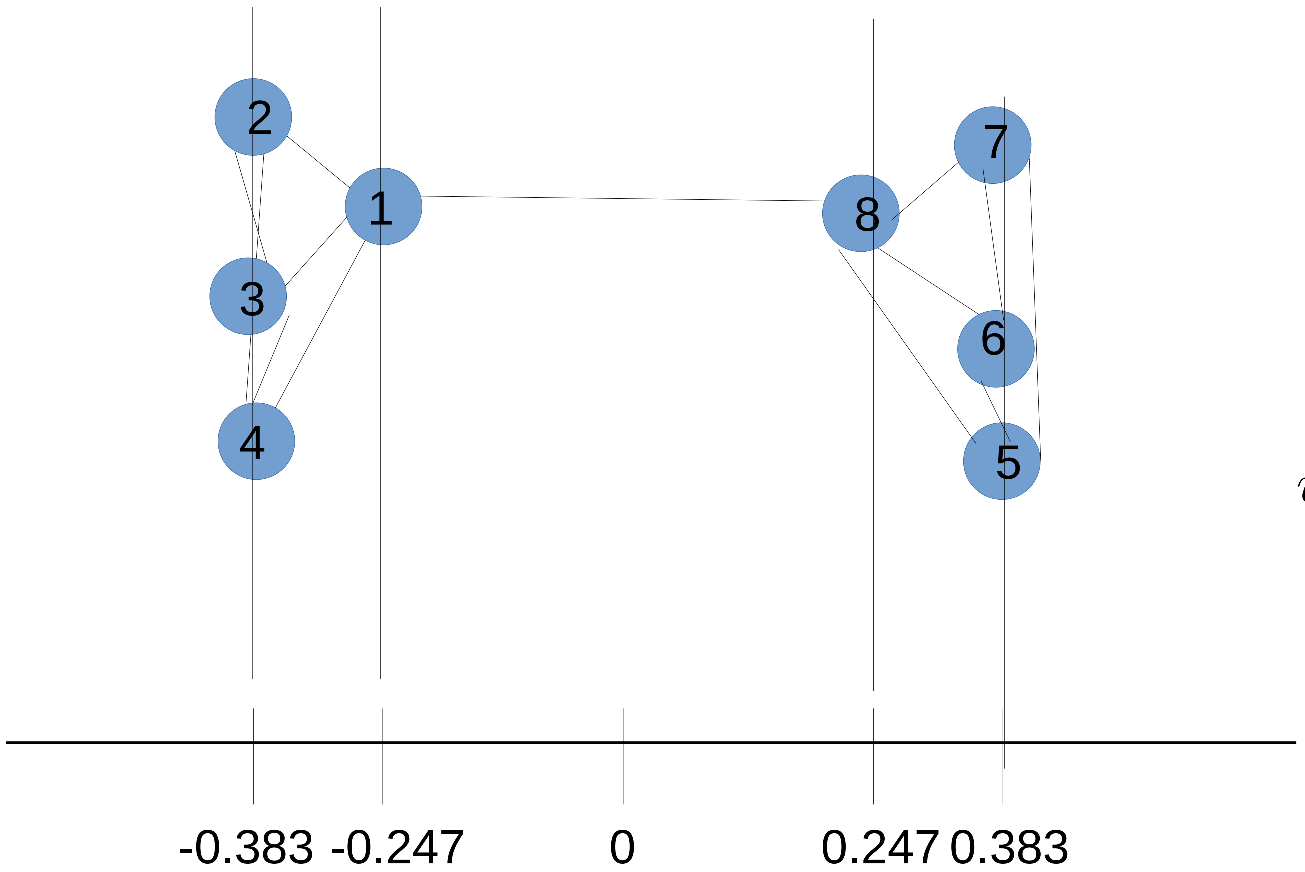




$$\lambda_1 = 0$$
$$\lambda_2 = 0.354$$

$$L = \begin{bmatrix} 4 & -1 & -1 & -1 & 0 & 0 & 0 & -1 \\ -1 & 3 & -1 & -1 & 0 & 0 & 0 & 0 \\ -1 & -1 & 3 & -1 & 0 & 0 & 0 & 0 \\ -1 & -1 & -1 & 3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 & -1 & -1 & -1 \\ 0 & 0 & 0 & 0 & -1 & 3 & -1 & -1 \\ 0 & 0 & 0 & 0 & -1 & -1 & 3 & -1 \\ -1 & 0 & 0 & 0 & -1 & -1 & -1 & 4 \end{bmatrix}$$

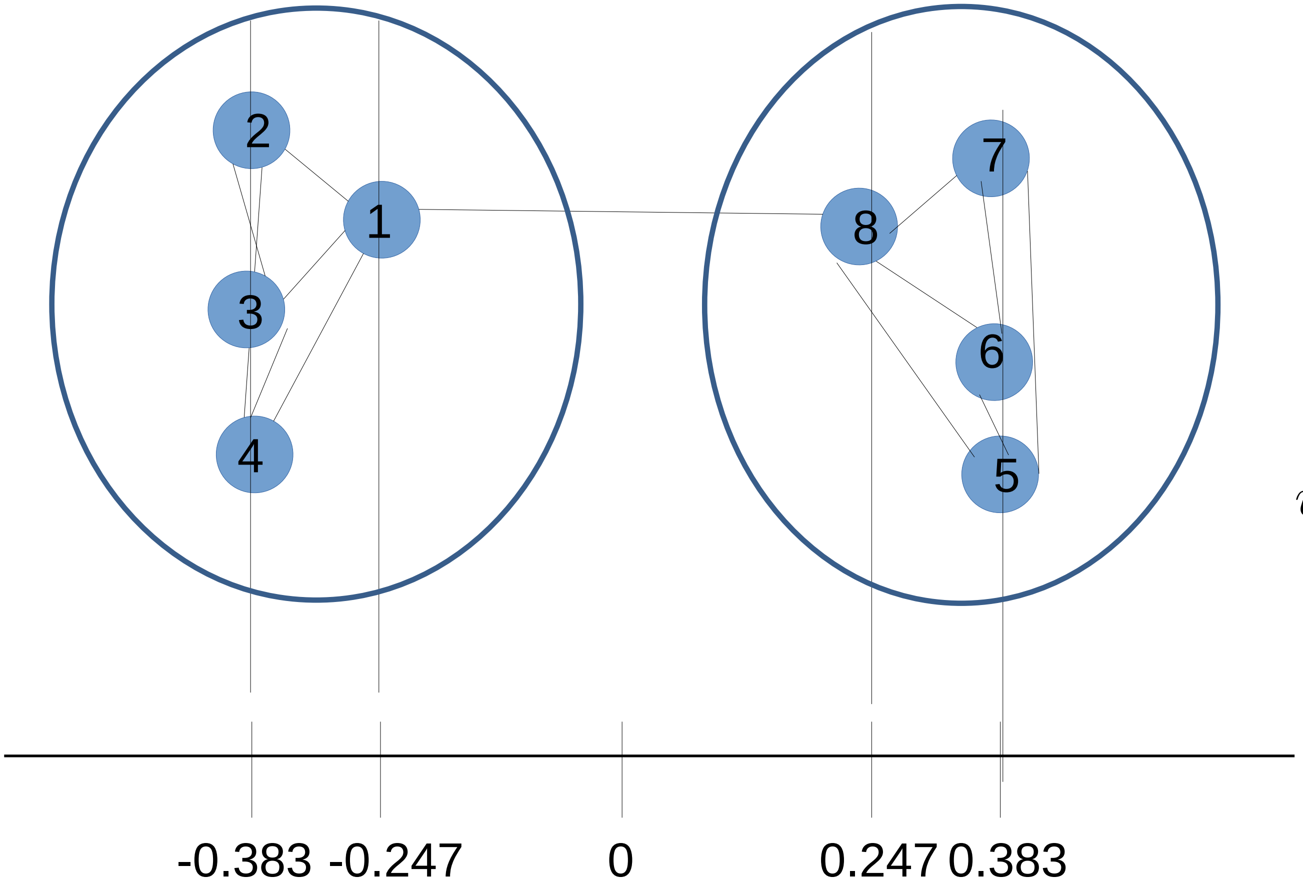
$$v_2 = \begin{bmatrix} 0.247 \\ 0.383 \\ 0.383 \\ 0.383 \\ -0.383 \\ -0.383 \\ -0.383 \\ -0.247 \end{bmatrix}$$



$$\lambda_1 = 0$$

$$\lambda_2 = 0.354$$

$$v_2 = \begin{bmatrix} 0.247 \\ 0.383 \\ 0.383 \\ 0.383 \\ -0.383 \\ -0.383 \\ -0.383 \\ -0.247 \end{bmatrix}$$

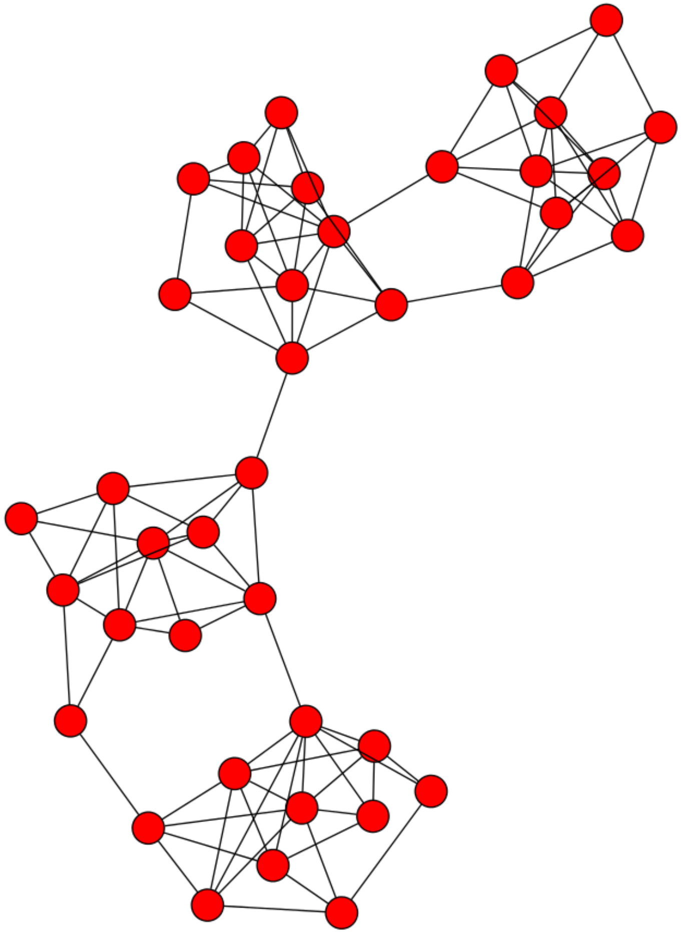


$$\lambda_2 = 0.354$$

$$v_2 = \begin{bmatrix} 0.247 \\ 0.383 \\ 0.383 \\ 0.383 \\ -0.383 \\ -0.383 \\ -0.383 \\ -0.247 \end{bmatrix}$$

k-Way Spectral Clustering

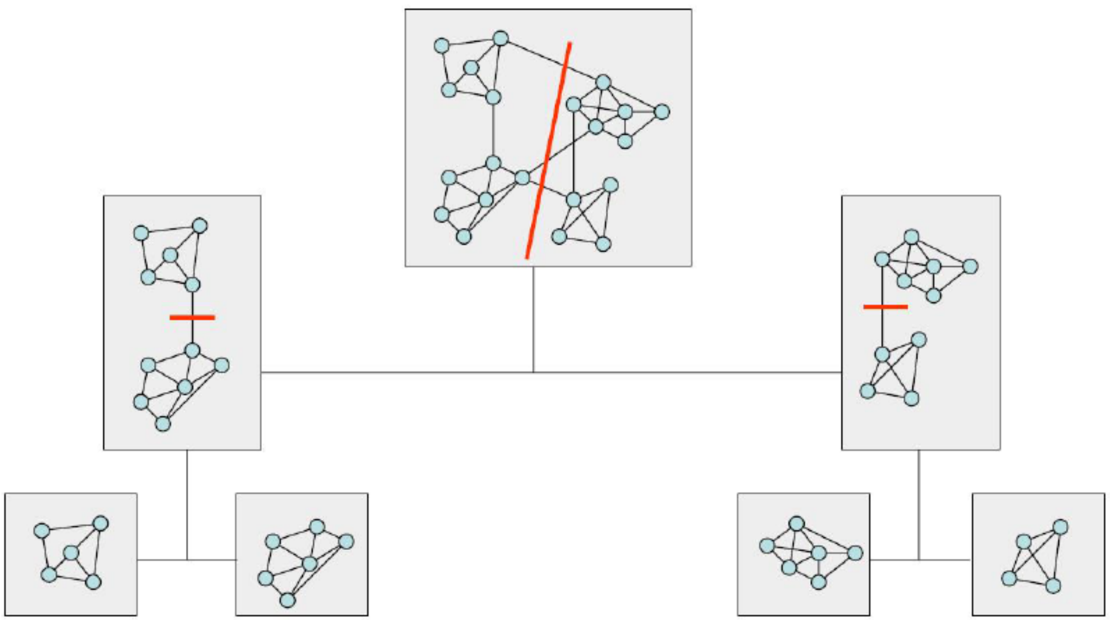
How do we partition a graph into k clusters?



k-Way Spectral Clustering

How do we partition a graph into k clusters?

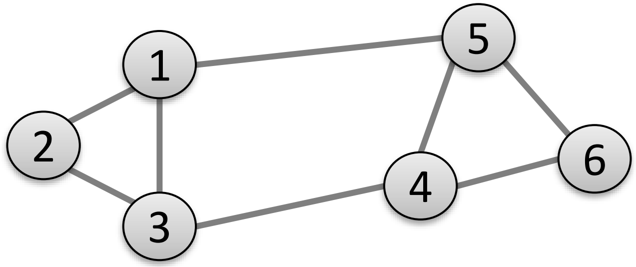
- *Recursively apply a bi-partitioning algorithm* in a hierarchical divisive manner
    - Disadvantages: Inefficient, unstable



k-Way Spectral Clustering

Use *several of the eigenvectors* to partition the graph.

- Use m eigenvectors, and set a threshold for each,
- Get a partition into 2^m groups, each group consisting of the nodes that are above or below threshold for each of the eigenvectors, in a particular pattern.



Example

Eigenvalues

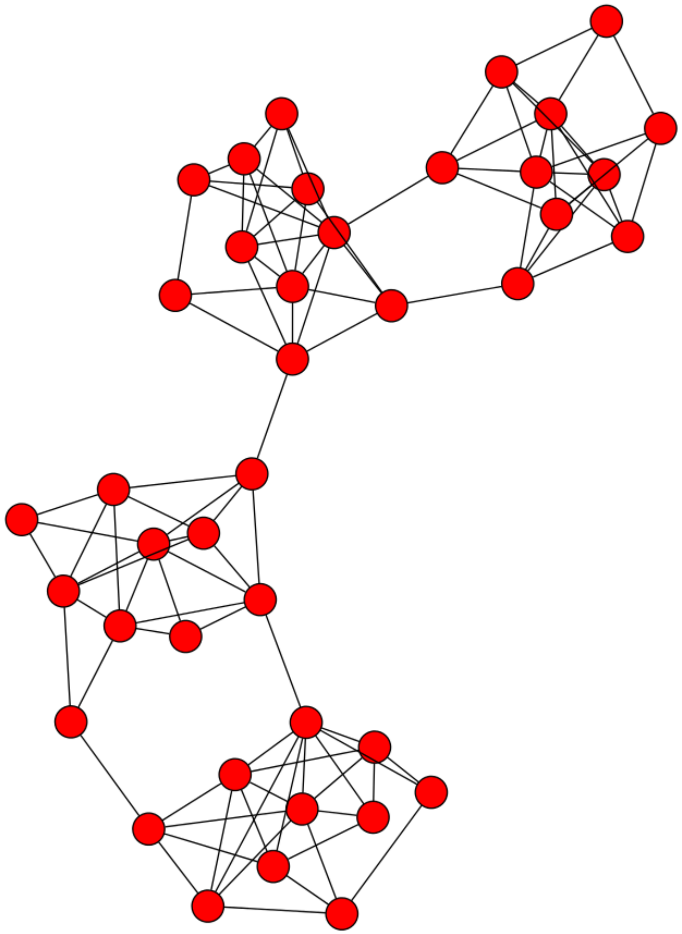
Eigenvectors

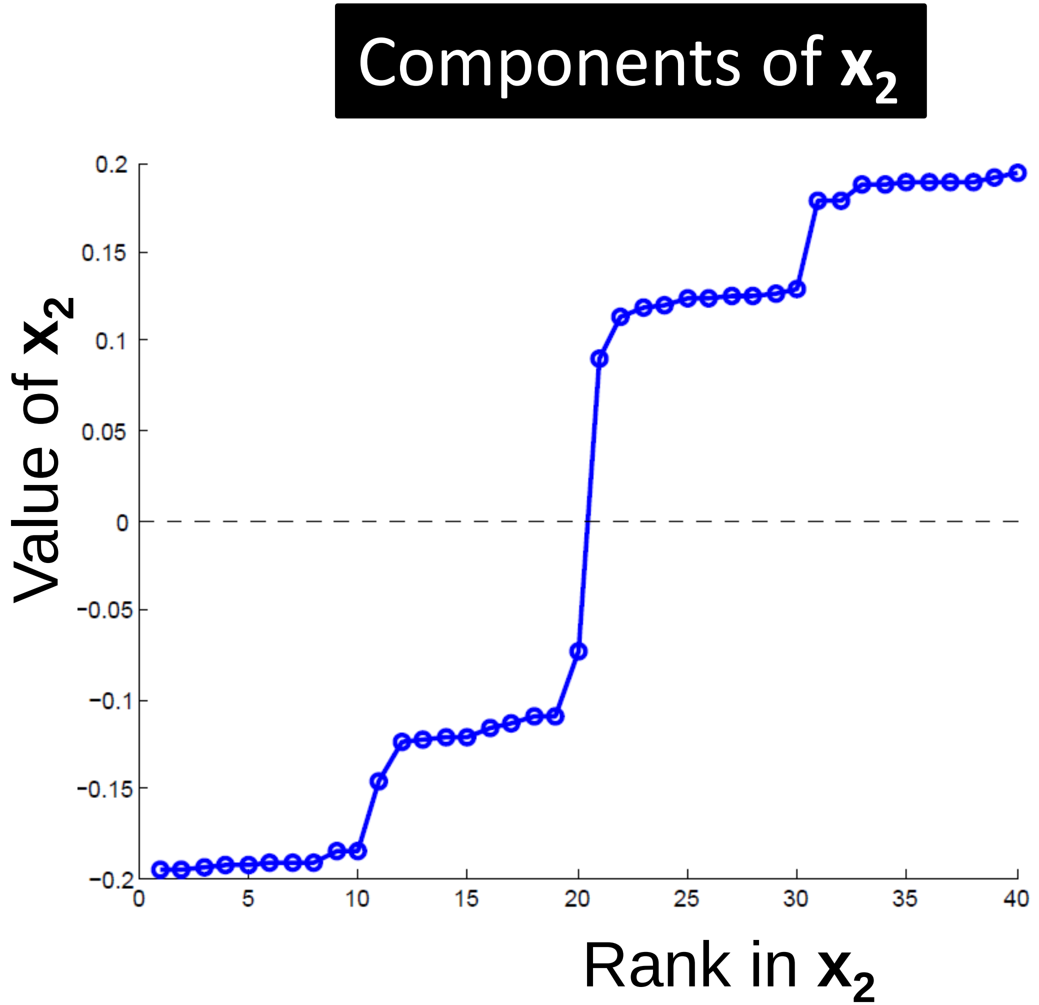
0.0	1.0	3.0	3..0	4..0	5.0
0.4	0.3	-0.5	-0.2	-0.4	-0.5
0.4	0.6	0.4	-0.4	0.4	0.0
0.4	0.3	0.1	0.6	-0.4	0.5
0.4	-0.3	0.1	0.6	0.4	-0.5
0.4	-0.3	-0.5	-0.2	0.4	0.5
0.4	-0.6	0.4	-0.4	-0.4	0.0

If we use both the 2nd and 3rd eigenvectors,
nodes 5 and 6 (negative in both) 2 and 3 (positive in both)
1 and 4 alone

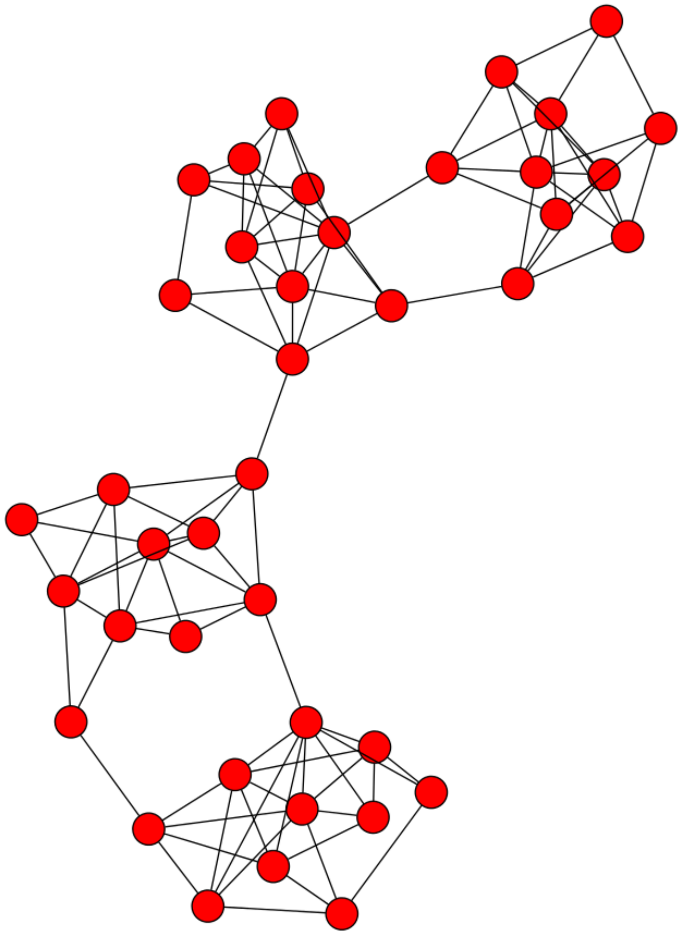
- Note that each eigenvector except the first is the vector x that minimizes $x^T Lx$, subject to the constraint that it is orthogonal to all previous eigenvectors.
- Thus, while each eigenvector tries to produce a minimum-sized cut, successive eigenvectors have to satisfy more and more constraints => the cuts progressively worse.

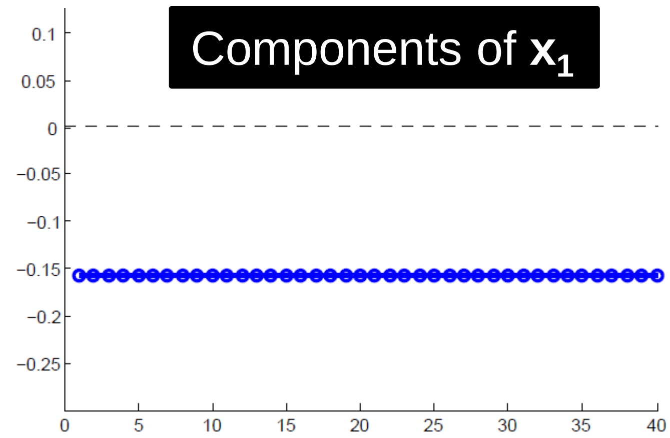
Example: Spectral Partitioning

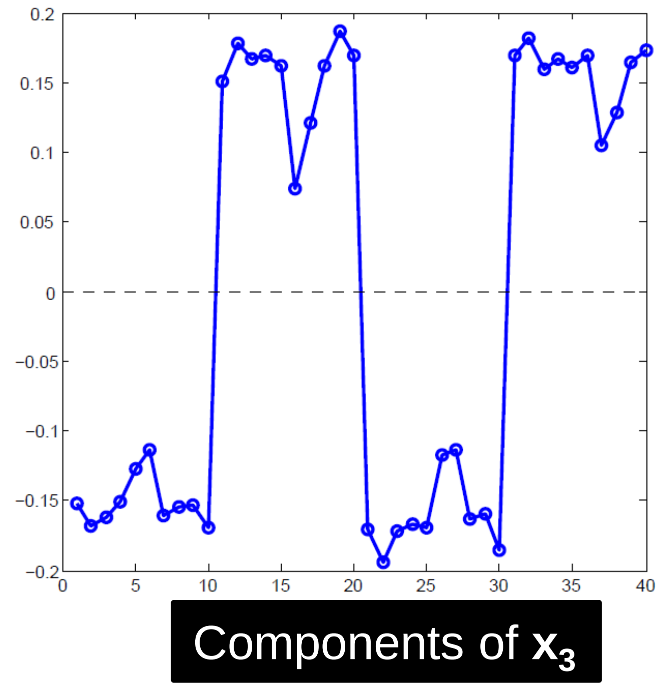




Example: Spectral partitioning







Spectral Clustering

- Use the lowest *k* eigenvalues of L to construct the *n* x *k* graph G' that has these eigenvectors as columns

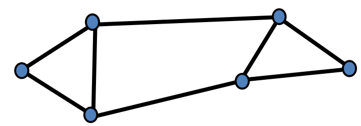
- *The n-rows represent the graph vertices in a k-dimensional Euclidean space*

- Group these vertices in *k* clusters using k-means clustering or similar techniques

k-Way Spectral Clustering

Pre-processing:

Build Laplacian matrix L of the graph



	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

Decomposition:

– Find eigenvalues λ and eigenvectors x of the matrix L

λ =

0.0
1.0
3.0
3.0
4.0
5.0

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

k = 3

Cuts and spectral clustering

$$\mathrm{cut}(A_1, \ldots, A_k) := \sum_{i=1}^{k} \mathrm{cut}(A_i, \overline{A}_i)$$

$$\mathrm{RatioCut}(A_1, \ldots, A_k) = \sum_{i=1}^{k} \frac{\mathrm{cut}(A_i, \overline{A}_i)}{|A_i|}$$

$$\mathrm{Ncut}(A_1, \ldots, A_k) = \sum_{i=1}^{k} \frac{\mathrm{cut}(A_i, \overline{A}_i)}{\mathrm{vol}(A_i)}.$$

Relaxing Ncut leads to normalized spectral clustering, while relaxing RatioCut leads to unnormalized spectral clustering

Normalized Graph Laplacians

$$L_{sym} = D^{-1/2} L D^{-1/2} = I - D^{-1/2} A D^{-1/2}$$

$$x^{\tau} L_{sym} x = \sum_{(i,j)\in E} \left(\frac{x_i}{\sqrt{d_i}} - \frac{x_j}{\sqrt{d_j}} \right)^2$$

$$L_{rw} = D^{-1} L = I - D^{-1} A$$

L_{rw} closely connected to random walks

Spectral clustering (besides graphs)

Can be used to cluster any points (not just vertices), as long as there is an appropriate similarity matrix

Needs to be *symmetric* and *non-negative*

How to construct a graph:

- ε-neighborhood graph: connect all points whose pairwise distances are smaller than ε
- k-nearest neighbor graph: connect each point with each k nearest neighbor
- full graph: connect all points with weight in the edge (i, j) equal to the similarity of i and j

Summary

- The values of x minimize

$$\min_{x \neq 0} \sum_{(i,j) \in E} \left(x_i - x_j \right)^2 \qquad \sum_i x_i = 0$$

- For weighted matrices

$$\min_{x \neq 0} \sum_{(i,j)} A[i,j] \left(x_i - x_j \right)^2 \qquad \sum_i x_i = 0$$

- The ordering according to the x_i values will group similar (connected) nodes together

Outline

PART II

Cuts

Spectral Clustering

Dense Subgraphs

Evaluation

Finding Dense Subgraphs

- Dense subgraph: A collection of vertices such that there are a lot of edges between them
  - E.g., find the subset of email users that talk the most between them
  - Or, find the subset of genes that are most commonly expressed together

- Similar to community identification but we do *not require* that the dense subgraph is *sparsely connected with the rest of the graph*.

Definitions

- Input: undirected graph $G = (V, E)$.
- Degree of node u: $\deg(u)$
- For two sets $S \subseteq V$ and $T \subseteq V$:

$$E(S,T) = \{(\mathrm{u},\mathrm{v}) \in E: u \in S, v \in T\}$$

- $E(S) = E(S,S)$: edges within nodes in S
- Graph Cut defined by nodes in $S \subseteq V$:

$E(S,\bar{S})$: edges between S and the rest of the graph

- Induced Subgraph by set S : $G_S = (S, E(S))$

Definitions

- How do we define the density of a subgraph?
- Average Degree:

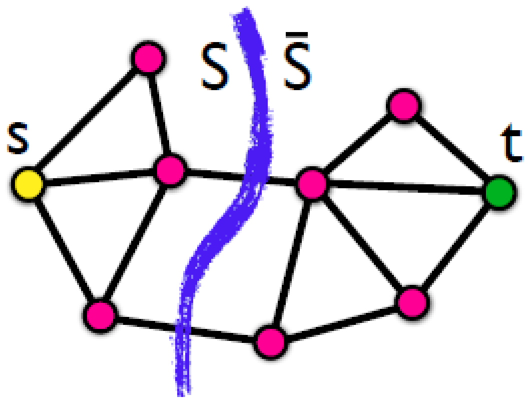
$$d(S) = \frac{2|E(S)|}{|S|}$$

- Problem: Given graph G, find subset S, that maximizes density d(S)
  - Surprisingly there is a polynomial-time algorithm for this problem.

The *k*-densest subgraph

- The *k*-densest subgraph problem: Find the set of k nodes S, such that the density $d(S)$ is maximized.
  - The k-densest subgraph problem is NP-hard!

Min-Cut Problem



Given a graph* $G = (V, E)$,
A source vertex $s \in V$,
A destination vertex $t \in V$

Find a set $S \subseteq V$
Such that $s \in S$ and $t \in \bar{S}$
That minimizes $E(S, \bar{S})$

* The graph may be weighted

Min-Cut = Max-Flow: the minimum cut maximizes the flow that can be sent from s to t. There is a polynomial time solution.

the *maximum amount of flow* passing from the source to the sink is equal to the total weight of the edges in the minimum cut

Algorithm (Goldberg)

Given the input graph G, and value c

1. Create the min-cut instance graph
2. Compute the min-cut
3. If the set S is not empty, return YES
4. Else return NO

How do we find the set with maximum density?

Min-cut algorithm

- The min-cut algorithm finds the optimal solution in polynomial time O(nm), but this is too expensive for real networks.
- We will now describe a simpler approximation algorithm that is very fast
  - Approximation algorithm: the ratio of the density of the set produced by our algorithm and that of the optimal is bounded.
    - The ratio is at most ½
    - The optimal set is at most twice as dense as that of the approximation algorithm.

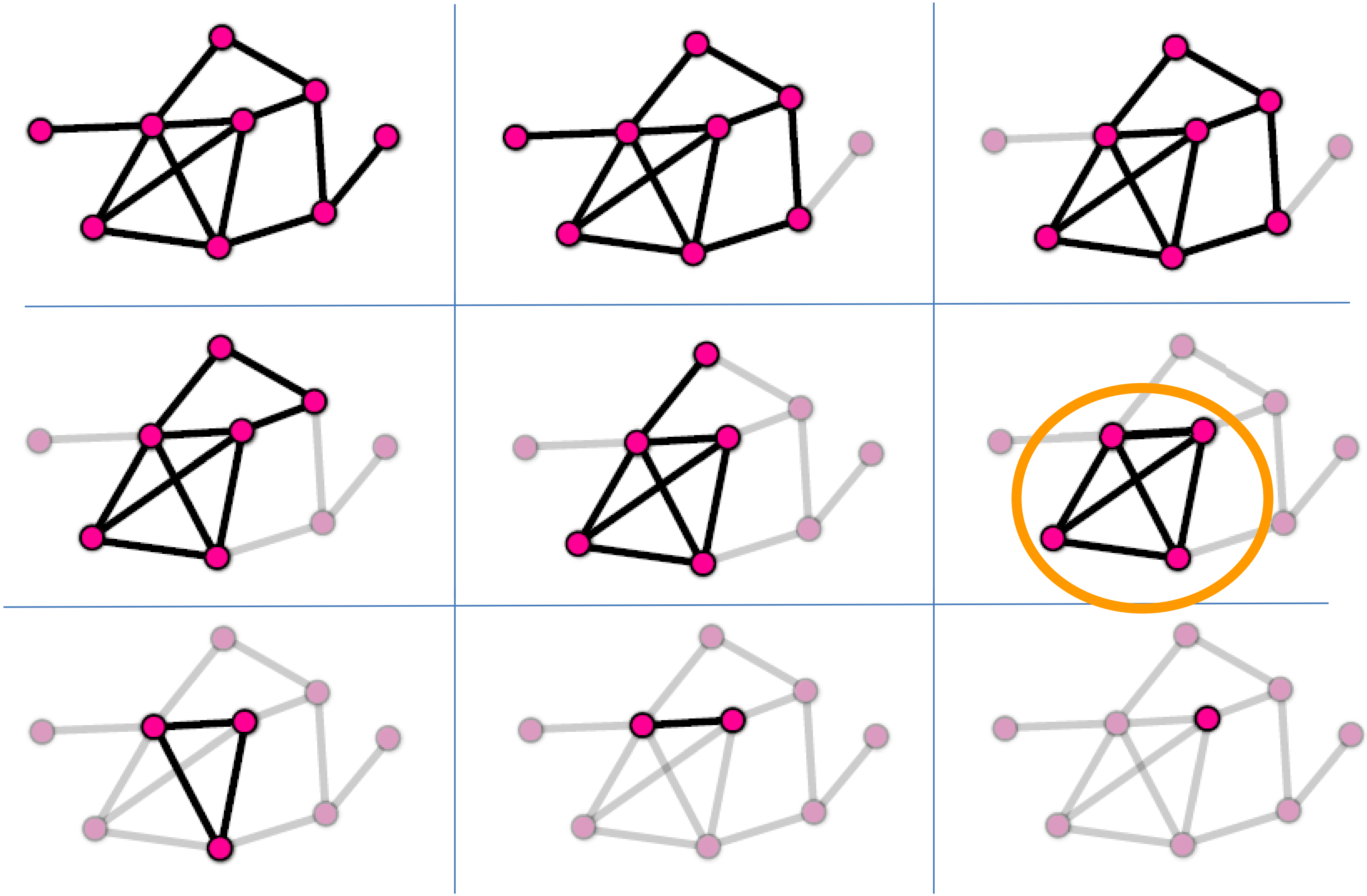
- Any ideas for the algorithm?

Greedy Algorithm

Given the graph $G = (V, E)$

1. $S_0 = V$
2. For $i = 1 \dots |V|$
   a. Find node $v \in S$ with the minimum degree
   b. $S_i = S_{i-1} \setminus \{v\}$
3. Output the densest set S_i

Example



Analysis

- Density of optimal set: $d_{opt} = \max_{S \subseteq V} d(S)$
- Density of greedy algorithm d_g

- $d_{opt} \leq 2 \cdot d_g$

Outline

PART II

Cuts

Spectral Clustering

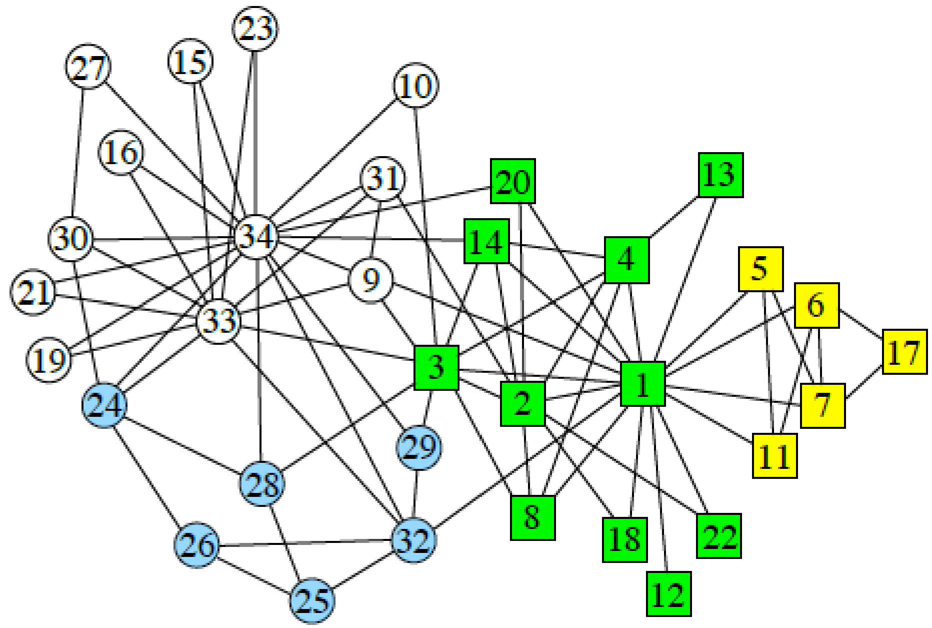
Dense Subgraphs

Evaluation

Community Evaluation

- With ground truth
- Without ground truth

Evaluation with ground truth



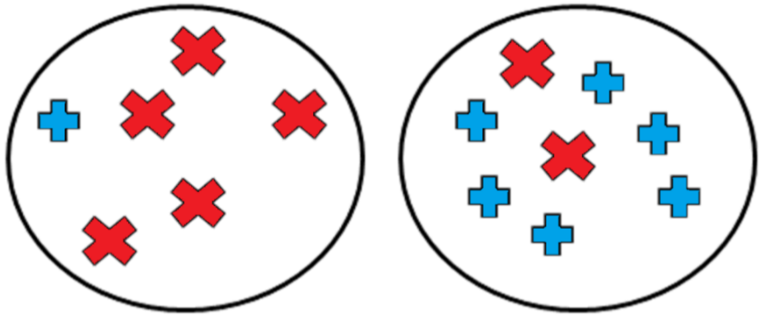
Zachary's Karate Club

Club president (34) (circles) and instructor (1) (rectangles)

Community Evaluation

We are given objects of two different kinds (+, ×)

- **The perfect community:** all objects inside the community are of the same type



Metrics: purity

the fraction of instances that have labels equal to the label of the community's majority

$$Purity = \frac{1}{N} \sum_{i=1}^{k} \max_{j} |C_i \cap L_j|$$

(5+6+4)/20 = 0.75

Purity can be easily **tampered** by

- Points being singleton communities (of size 1); or by
- Very large communities

Metrics

Based on pair counting: the number of pairs of vertices which are classified in the same (different) clusters in the two partitions.

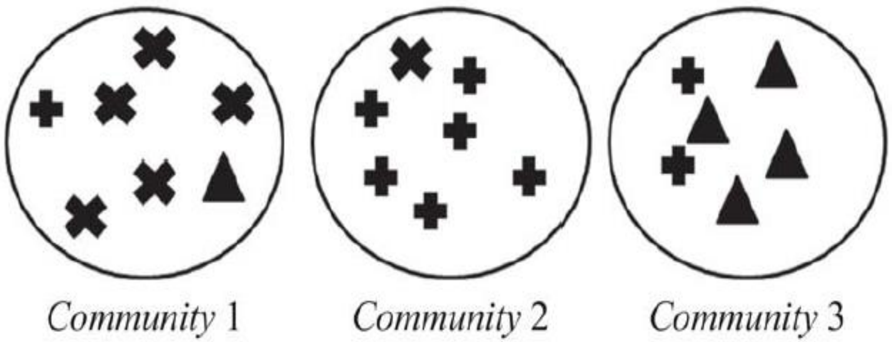
- True Positive (**TP**) Assignment: when *similar members* are assigned to the *same community*. This is a correct decision.
- True Negative (**TN**) Assignment: when *dissimilar members* are assigned to *different communities*. This is a correct decision.
- False Negative (**FN**) Assignment: when *similar members* are assigned to *different communities*. This is an incorrect decision.
- False Positive (**FP**) Assignment: when *dissimilar members* are assigned to the *same community*. This is an incorrect decision.

Metrics: pairs

For TP, we need to compute the number of pairs with the same label that are in the same community

$$TP = \underbrace{\binom{5}{2}}_{Community\ 1} + \underbrace{\binom{6}{2}}_{Community\ 2} + \underbrace{\left(\binom{4}{2} + \binom{2}{2}\right)}_{Community\ 3} = 32$$

Metrics: pairs



$$
\begin{aligned}
TN \;=\; & \underbrace{(\overbrace{5\times 6}^{\times,+} + \overbrace{1\times 1}^{+,\times} + \overbrace{1\times 6}^{\triangle,+} + \overbrace{1\times 1}^{\triangle,\times})}_{\textit{Communities 1 and 2}} \\
& + \underbrace{(\overbrace{5\times 4}^{\times,\triangle} + \overbrace{5\times 2}^{\times,+} + \overbrace{1\times 4}^{+,\triangle} + \overbrace{1\times 2}^{\triangle,+})}_{\textit{Communities 1 and 3}} \\
& + \underbrace{(\overbrace{6\times 4}^{+,\triangle} + \overbrace{1\times 2}^{\times,+} + \overbrace{1\times 4}^{\times,\triangle}}_{\textit{Communities 2 and 3}} = 104.
\end{aligned}
$$

For TN: we need to compute the number of dissimilar pairs in dissimilar communities

Metrics: pairs

For FP, compute dissimilar pairs that are in the same community.

$$FP = \underbrace{(5 \times 1 + 5 \times 1 + 1 \times 1)}_{Community\ 1} + \underbrace{(6 \times 1)}_{Community\ 2} + \underbrace{(4 \times 2)}_{Community\ 3} = 25$$

For FN, compute similar members that are in different communities.

$$FN = \underbrace{(5 \times 1)}_{\times} + \underbrace{(6 \times 1 + 6 \times 2 + 2 \times 1)}_{+} + \underbrace{(4 \times 1)}_{\triangle} = 29$$

Metrics: pairs

Precision (P): the fraction of pairs that have been correctly assigned to the same community.

$$TP/(TP+FP)$$

Recall (R): the fraction of pairs assigned to the same community of all the pairs that should have been in the same community.

$$TP/(TP+FN)$$

F-measure

$$2PR/(P+R)$$

Evaluation without ground truth

- Cluster Cohesion: Measures how closely related are objects in a cluster
- Cluster Separation: Measure how distinct or well-separated a cluster is from other clusters
- Example: Squared Error
  - Cohesion is measured by the within cluster sum of squares (SSE)

$$WSS = \sum_{i} \sum_{x \in C_i} (x - m_i)^2$$

  - Separation is measured by the between cluster sum of squares

$$BSS = \sum_{i} |C_i| (m - m_i)^2$$

  - Where $|C_i|$ is the size of cluster i

Evaluation without ground truth

$$cohesion(C) = \frac{\#internal\ edges\ of\ C}{N_C(N_C - 1)/2}$$

$$seperation(C) = \frac{\#inter - cluster\ edges\ of\ C}{N_C(N - N_C)}$$

Evaluation without ground truth

Modularity

Both as a local (per individual community) and as a global measure

Conductance

Evaluation without ground truth

With semantics:

- (ad hoc) analyze other attributes (e.g., profile, content generated) for coherence
- human subjects (user study) Mechanical Turk

Visual representation (similarity/adjacency matric, word clouds, etc)

(a) U.S . Constitution

(b) Sports

Summary

- Spectral clustering

Using the eigenvectors of the Laplacian (or, normalized Laplacian)

split around 0

use the k-eigenvectors

- Dense subgraphs

- Evaluation

Basic References

- Jure Leskovec, Anand Rajaraman, Jeff Ullman, Mining of Massive Datasets, Chapter 10, http://www.mmds.org/
- Reza Zafarani, Mohammad Ali Abbasi, Huan Liu, Social Media Mining: An Introduction, Chapter 6, http://www.socialmediamining.info/
- Santo Fortunato: Community detection in graphs. CoRR abs/0906.0612v2 (2010)
- Ulrike von Luxburg: A Tutorial on Spectral Clustering. CoRR abs/0711.0189 (2007)
- Albert-László Barabasi, Network Science, Chapter 9, http://networksciencebook.com/

Questions?

Not used

Graph partitioning

The general problem

- Input: a graph $G = (V, E)$
  - edge (u, v) denotes connection/similarity between u and v
  - weighted graphs: *weight* of edge captures the degree of similarity (or, strength of connection)

Partition the nodes in the graph such that

- nodes *within clusters* are *well interconnected* (high edge weights)
- nodes *across clusters* are *sparsely interconnected* (low edge weights)

Partitioning as an optimization problem:

- most graph partitioning problems are NP hard

Reformulating ratio cut (or normalized cut) in matrix format

- Let $X_{ij} = 1$, when node i is member of community j; 0, otherwise
- Let $D = diag(d_1, d_2, \cdots, d_n)$ be the diagonal degree matrix
- The ith entry on the diagonal of $X^T AX$ is the number of edges that are inside community i.
- The ith element on the diagonal of $X^T DX$ is the number of edges that are connected to members of community i.
- The ith element on the diagonal of $X^T(D - A)X$ is the number of edges in the cut that separates community i from other nodes.

The ith diagonal element of $X^T(D - A)X$ is equivalent to cut$(P_i, \bar{P}_i)$

So, ratio cut is

$$\begin{aligned}
\text{Ratio Cut}(P) &= \frac{1}{k}\sum_{i=1}^{k}\frac{\text{cut}(P_i,\bar{P}_i)}{|P_i|} \\
&= \frac{1}{k}\sum_{i=1}^{k}\frac{X_i^T(D-A)X_i}{X_i^T X_i} \\
&= \frac{1}{k}\sum_{i=1}^{k}\hat{X}_i^T(D-A)\boxed{\hat{X}_i}
\end{aligned}$$

$$\hat{X}_i = \hat{X}_i/(\hat{X}_i^T\hat{X}_i)^{1/2}$$

Both ratio/normalized cut can be reformulated as

$$\min_{\hat{X}} \operatorname{Tr}(\hat{X}^T L \hat{X})$$

$$L = \begin{cases} D - A & \text{Ratio Cut Laplacian, i.e., Unnormalized Laplacian} \\ I - D^{-1/2} A D^{-1/2} & \text{Normalized Laplacian for Normalized Cut.} \end{cases}$$

$D = diag(d_1, d_2, \cdots, d_n)$ is a diagonal degree matrix

- Spectral relaxation:

$$\min_{\hat{X}} \operatorname{Tr}(\hat{X}^T L \hat{X})$$
$$s.t.\ \hat{X}^T \hat{X} = I_k$$

Optimal Solution

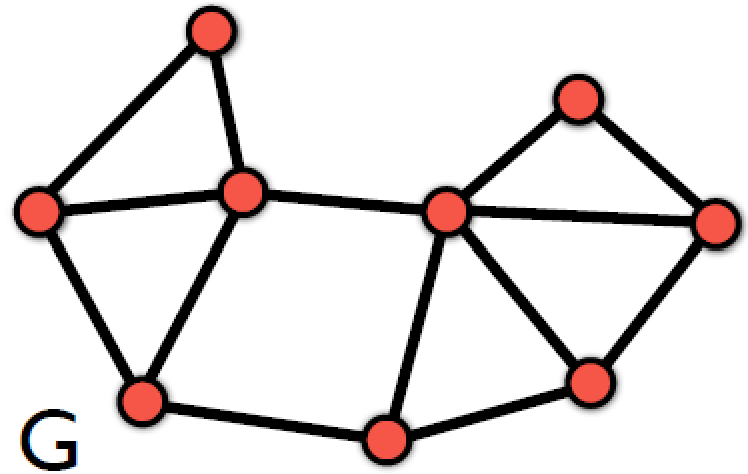
$\hat{X}$ is the top eigenvectors with the smallest eigenvalues

- Because we performed spectral relaxation, the matrix obtained is not integer valued

- To recover X from $\hat{X}$ we can run k-means on $\hat{X}$

Decision problem

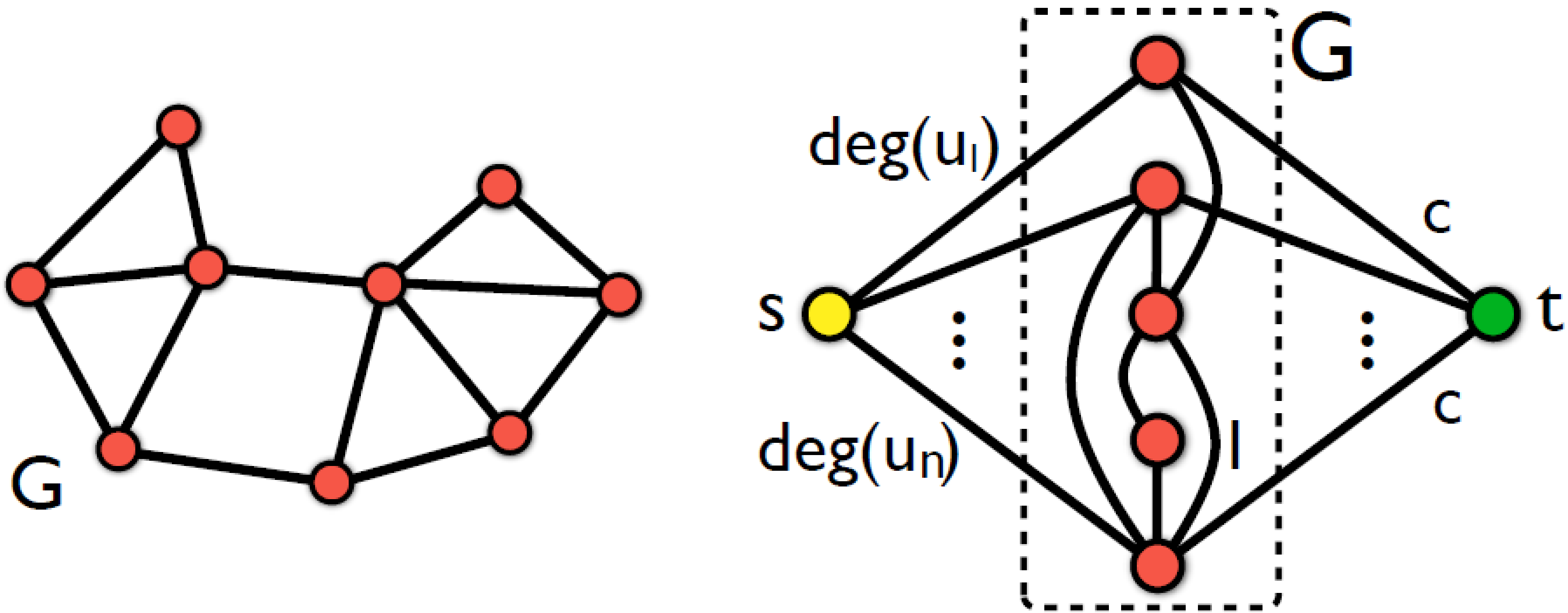
- Consider the decision problem:
  - Is there a set S with $d(S) \geq c$?



- $d(S) \geq c$
- $2|E(S)| \geq c|S|$
- $\sum_{v \in S} \deg(v) - E(S, \bar{S}) \geq c|S|$
- $2|E| - \sum_{v \in \bar{S}} \deg(v) - E(S, \bar{S}) \geq c|S|$
- $\sum_{v \in \bar{S}} \deg(v) + E(S, \bar{S}) + c|S| \leq 2|E|$

Transform to min-cut

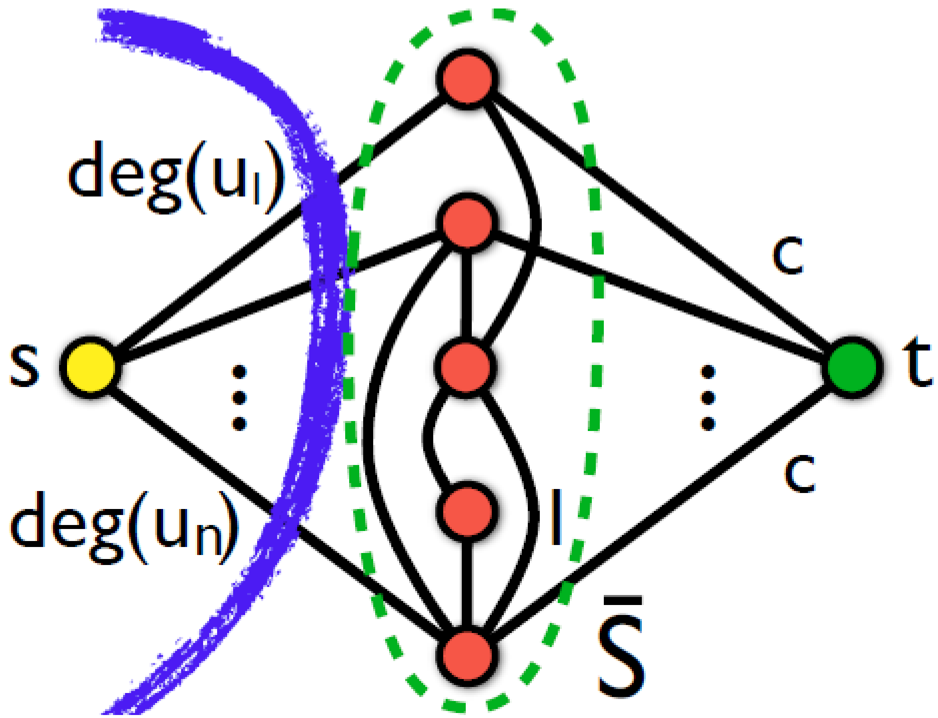
- For a value c we do the following transformation



- We ask for a min s-t cut in the new graph

Transformation to min-cut

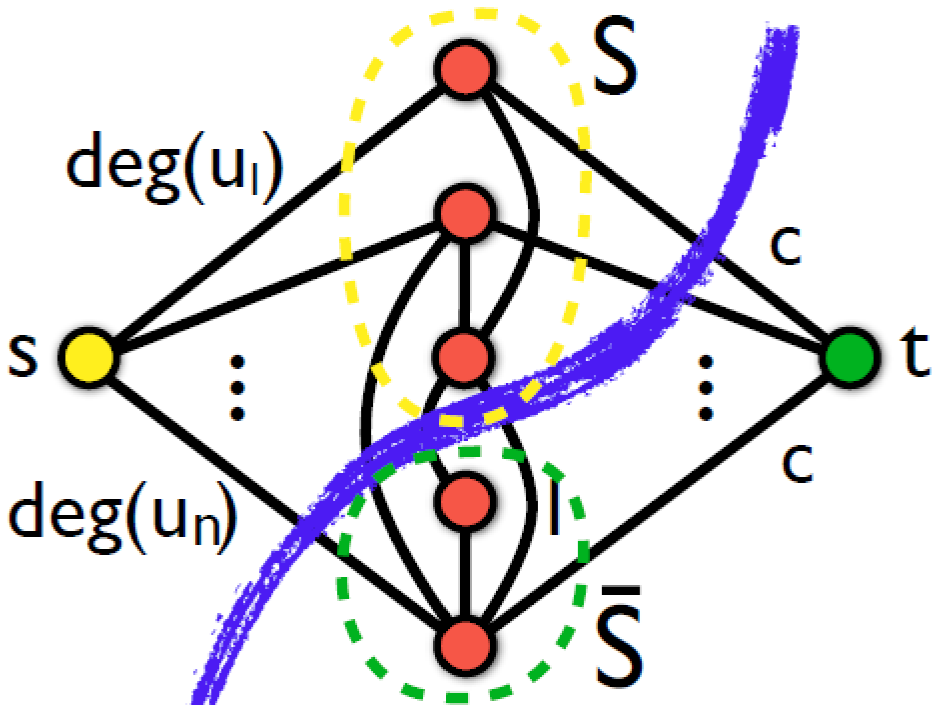
- There is a cut that has value $2|E|$



Transformation to min-cut

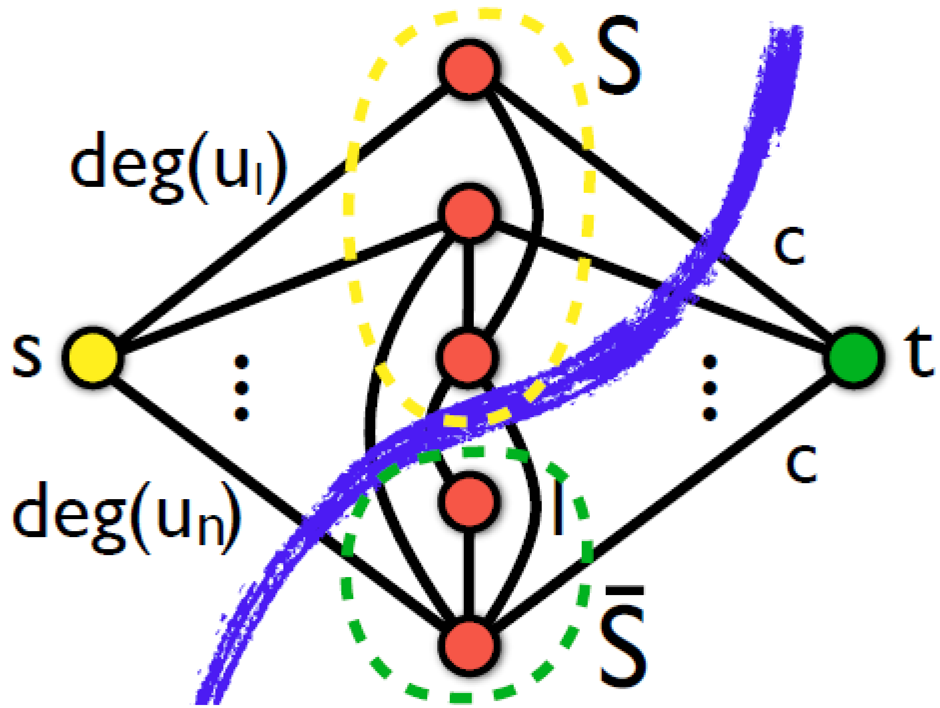
Every other cut has value:

- $\sum_{v \in \bar{S}} \deg(v) + E(S, \bar{S}) + c|S|$



Transformation to min-cut

- If $\sum_{v \in \bar{S}} \deg(v) + E(S, \bar{S}) + c|S| \leq 2|E|$ then $S \neq \emptyset$ and $d(S) \geq c$



Analysis

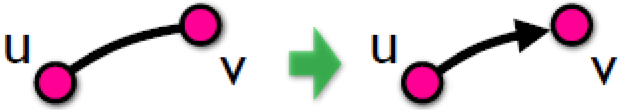
- We will prove that the optimal set has density at most 2 times that of the set produced by the Greedy algorithm.

- Density of optimal set: $d_{opt} = \max_{S \subseteq V} d(S)$
- Density of greedy algorithm d_g

- We want to show that $d_{opt} \leq 2 \cdot d_g$

Upper bound

- We will first upper-bound the solution of optimal
- Assume an arbitrary assignment of an edge (u, v) to either u or v



- Define:
  - $IN(u)$= # edges assigned to u
  - $\Delta = \max_{u \in V} IN(u)$
- We can prove that
  - $d_{opt} \leq 2 \cdot \Delta$

This is true for any assignment of the edges!

Lower bound

- We will now prove a lower bound for the density of the set produced by the greedy algorithm.
- For the lower bound we consider a specific assignment of the edges that we create as the greedy algorithm progresses:
  - When removing node u from S, assign all the edges to u
- So: $IN(u) = \text{degree of } u \text{ in } S \leq d(S) \leq d_g$
- This is true for all u so $\Delta \leq d_g$

- It follows that $d_{opt} \leq 2 \cdot d_g$