Chapter 8-2: Community Detection

Jilles Vreeken
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

Use this to define “topics”: What the same people on the left talk about on the right

Dense 2-layer graph
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”

\[ |X| = \alpha = 3 \]
\[ |Y| = \beta = 4 \]

\[ K_{3,4} \]
Frequent itemset mining

Recall market basket analysis.

- market: universe $U$ of $n$ items
- baskets: $m$ transactions, 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 $t_i \in D$

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

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

$\alpha$ … 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 $t_i$
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
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 $C_1, C_2, ..., C_k$ 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 connected to $A$:

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

where $w_{ij}$ is the weight of edge $(i, j)$.

In normalized cut, we measure the size of $C_i$ instead of $C_i$ itself:

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

Finding the optimal RatioCut or NormalisedCut is **NP-hard**.
Spectral Graph Partitioning

- \( A \): adjacency matrix of undirected \( G \)
  - \( A_{ij} = 1 \) if \( i \) is an edge, else 0
- \( x \) 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 \( A \cdot x \)?

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

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

Entry \( y_i \) is a sum of labels \( x_j \) of neighbors of \( i \)
What is the meaning of $Ax$?

The $j^{th}$ coordinate of $A \cdot x$:
- sum of the $x$-values of neighbors of $j$
- make this a new value at node $j$

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

$A \cdot x = \lambda \cdot x$

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 $G$?

$$A \cdot x = \lambda \cdot x$$

What is $\lambda$? What is $x$?

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

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

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

$$y_i = \sum_{j} A_{ij} x_j = \sum_{(i,j) \in E} x_j$$
Example: Graph of 2 components

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

What are some eigenvectors?
- $x = \text{put all 1s on } C_1 \text{ and 0s on } C_2 \text{ or vice versa}$
  - $x' = (1, \ldots, 1, 0, \ldots, 0)$ then $A \cdot x' = (d, \ldots, d, 0, \ldots, 0)$
  - $x'' = (0, \ldots, 0, 1, \ldots, 0)$ then $A \cdot x'' = (0, \ldots, 0, d, \ldots, d)$
  - and so, in both cases the corresponding $\lambda = d$

A bit of intuition:

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

If the graph is connected (right) then we already know that \( x_n = (1, ..., 1) \) is an eigenvector

Since eigenvectors are orthogonal, the components of \( x_{n-1} \) sum to 0

why? Because \( x_n \cdot x_{n-1} = \sum_i x_n[i] \cdot x_{n-1}[i] \)

So, we can look at the eigenvectors of the 2\textsuperscript{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 $A$ has the weight of edge $(i, j)$ at position $a_{ij}$

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

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 $\Delta_{ii} = d_i$

- $\Delta_{ii} = d_i = \sum_j a_{ij} = \text{degree of node } i$
- $n \times n$ diagonal matrix
Matrix representations (3)

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

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

(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

![Graph](image)

**Important properties:**

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

$$L = D - A$$

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>-1</td>
<td>2</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>-1</td>
<td>-1</td>
<td>3</td>
<td>-1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>3</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>5</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>3</td>
<td>-1</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>2</td>
</tr>
</tbody>
</table>
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

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

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 matrix $L^s$ or $\Lambda^s$ of a graph is defined as

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

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 $c_i$
- clusters are disjoint so $c_i^T c_j = 0$
- cluster has $c_i^T c_i = ||c_i||^2$ elements

We can get the $\text{vol}(C_i)$ and $W(C_i, V)$ using $c_i$’s
- $\text{vol}(C_i) = \sum_{j \in C_i} d_j = \sum_{r=1}^n \sum_{s=1}^n c_{ir} \Delta_{rs} c_{is} = c_i^T \Delta c_i$
- $W(C_i, C_i) = \sum_{r \in C_i} \sum_{s \in C_i} a_{rs} = c_i^T Ac_i$
- $W(C_i, V \setminus C_i) = W(C_i, V) - W(C_i, C_i) = c_i^T (\Delta - A)c_i = c_i^T Lc_i$
Cuts using matrices

\[
\text{RatioCut} = \sum_{i=1}^{k} \frac{W(C_i, V \setminus C_i)}{|C_i|} = \sum_{i=1}^{k} \frac{c_i^T L c_i}{|c_i|^2}
\]

\[
\text{NormalisedCut} = \sum_{i=1}^{k} \frac{W(C_i, V \setminus C_i)}{vol(C_i)} = \sum_{i=1}^{k} \frac{c_i^T L c_i}{c_i^T \Delta c_i}
\]
The second eigenvalue, $\lambda_2$, as an optimization problem

Fact: for symmetric matrix $M$:

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

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

$$x^T L x = \sum_{ij} L_{ij} x_i x_j = \sum_{ij} (D_{ij} - A_{ij}) x_i x_j$$

$$= \sum_i D_{ii} x_i^2 - \sum_{(i,j) \in E} 2 x_i x_j$$

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

Node $i$ has degree $d_i$. So, value $x_i^2$ needs to be summed up $d_i$ times. But each edge $(i, j)$ has two endpoints, so we need $x_i^2 + x_j^2$.
\( \lambda_2 \), as an optimization problem

What else do we know about \( x \)?

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

Remember

\[
\lambda_2 = \min_{\text{all labelings of nodes } i \text{ so that } \sum x_i = 0} \frac{\sum_{(i,j) \in E} (x_i - x_j)^2}{\sum_i x_i^2}
\]

We want to assign values \( x_i \) to nodes \( i \) such that few edges cross 0. (we want \( x_i \) and \( x_j \) to subtract each other)
Finding an optimal cut

Back to finding the optimal cut

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

\[ c_i = \begin{cases} +1 & \text{if } i \in C_1 \\ -1 & \text{if } i \in C_2 \end{cases} \]

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

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

NP-hard... so, let’s relax!
- let \(c_i\)’s take any real value

(Fiedler, 1973)
Rayleigh theorem

\[
\min_{c \in \mathbb{R}^n} f(c) = \sum_{(i,j) \in E} (c_i - c_j)^2 = c^T L c
\]

\[\lambda_2 = \min_c f(c): \text{the minimum value of } f(c) \text{ is given by the } 2^{nd} \text{ smallest eigenvalue } \lambda_2 \text{ of the Laplacian matrix } L\]

\[x = \arg\min_c f(c): \text{the optimal solution for } c \text{ is given by the corresponding eigenvector } x \text{ and is referred to as the Fiedler vector}\]
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 $L$ of the graph

2) Decomposition:
   - find eigenvalues $\lambda$ and eigenvectors $x$ of the matrix $L$
   - map vertices to corresponding components of $\lambda_2$

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)

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.3</td>
<td>0.6</td>
<td>0.3</td>
<td>-0.3</td>
<td>-0.3</td>
<td>-0.6</td>
</tr>
</tbody>
</table>

**Split at 0:**
- **Cluster A:** Positive points
- **Cluster B:** Negative points

A | B
---|---
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 $k$ 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 $O(n)$

\[
\begin{align*}
&v_1 & v_2 & \ldots & v_{k-1} & v_k & \ldots & v_{2k-1} & v_{2k} \\
&v_{2k+1} & v_{2k+2} & \ldots & v_{3k-1} & v_{3k} & \ldots & v_{4k-1} & v_{4k}
\end{align*}
\]
To do the clustering, we need to move our real-valued eigenvectors $u_i$ to binary cluster indicator vectors

First, create a matrix $U$ with $u_i$'s as its columns
- optionally, normalize the rows to sum up to 1 (esp when using $L^S$)

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 $U$ matrix takes $O(tnk^2)$
  - $t$ is the number of iterations in $k$-means
Another look at Approximate cuts

Allowing for real-valued cluster assignment vectors \( c_i \) makes Relaxed RatioCut look like

\[
J_{rc}(C) = \sum_{i=1}^{k} \frac{c_i^T L c_i}{\|c_i\|^2} = \sum_{i=1}^{k} \left( \frac{c_i}{\|c_i\|} \right)^T L \left( \frac{c_i}{\|c_i\|} \right) = \sum_{i=1}^{k} u_i^T L u_i
\]

\[ u_i = \frac{c_i}{\|c_i\|} \] i.e. the unit vector in the direction of \( c_i \)
Solving the relaxed version

We want to minimize the function $J_{rc}$ over $u_i$'s

- we have a constraint that $u_i^T u_i = 1$

To solve, derive w.r.t. $u_i$'s and find the roots

- add Lagrange multipliers to incorporate the constraints:

$$\frac{\delta}{\delta u_i} \left( \sum_{i}^k u_i^T L u_i + \sum_{i}^k \lambda_i (1 - u_i^T u_i) \right) = 0$$

Hence, $Lu_i = \lambda_i u_i$

- $u_i$ is an eigenvector of $L$ corresponding to the eigenvalue $\lambda_i$
Which eigenvectors to choose

We know that $Lu_i = \lambda_i u_i$
- hence $\lambda_i = u_i^T Lu_i$

As we’re minimizing the sum of $u_i^T Lu_i$’s we should choose the $u_i$’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 $(n^{1/2}, n^{1/2}, ..., n^{1/2})$
- hmm, that doesn’t help with clustering...
Normalised cut and choice of Laplacians

For normalized cut similar procedure shows that we should select the $k$ smallest eigenvectors of $L^s$ instead of $L$
- or, we can use the asymmetric Laplacian $L^a$

Which one we should choose?
- both ratio and normalised cut aim at minimising intra-cluster similarity
- only normalised cut considers inter-cluster similarity $\rightarrow$ either $L^s$ or $L^a$

The asymmetric Laplacian is better
- with symmetric one further normalisation is needed
Algorithm \textsc{SpectralClustering}(connected graph $G, k$) :
compute the similarity matrix $A \in \mathbb{R}^{n \times n}$ for $G$

if \textit{ratio cut} then $B \leftarrow L$
else if \textit{normalised cut} then $B \leftarrow L_s$ or $L^a$
solve $Bu_i = \lambda_i u_i$ for $i = k + 1$, where $\lambda_2 \leq \lambda_3 \leq \ldots \leq \lambda_{k+1}$
$U \leftarrow (u_n \ u_{n-1} \ldots u_{n-k+1})$
$Y \leftarrow$ normalise rows of $U$
$C \leftarrow \{C_1, \ldots , C_k\}$ via $k$-means on $Y$
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
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