Chapter X: Graph Mining

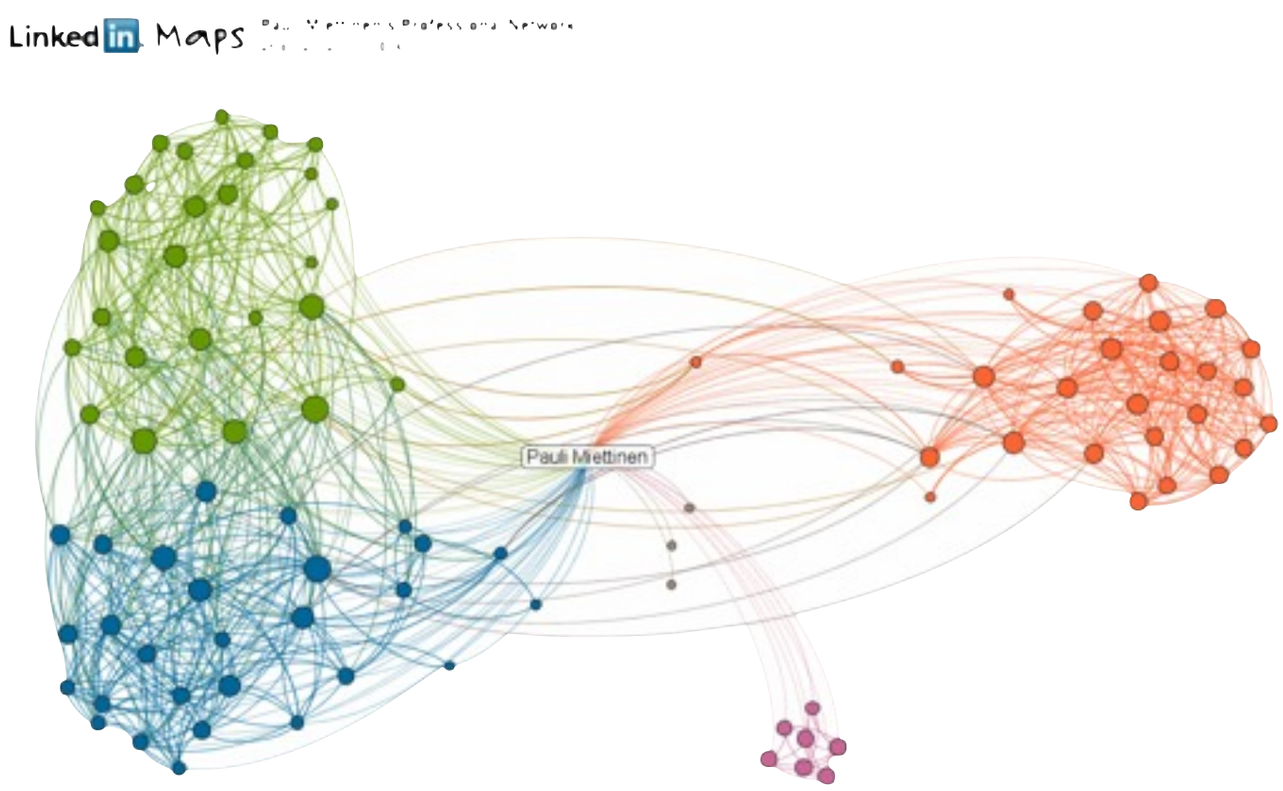
ZM Ch. 4, 11, 16

Chapter X.4: Graph Clustering

ZM Ch. 16, von Luxburg: *A tutorial on spectral clustering*, 2007

Where do Graphs Come From?

- We can have data in a graph form
  - E.g. the clusters of our social networks
- Or we can 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 pairs of elements that are very dissimilar
- We can remove the edges between dissimilar pairs of vertices
  - Zero weight
- Alternatively, we can adjust the weights to diminish dissimilar points
  - The **Gaussian kernel** is popular for this

$$w_{ij} = \exp\left\{-\frac{|\boldsymbol{x}_i - \boldsymbol{x}_j|^2}{2\sigma^2}\right\}$$

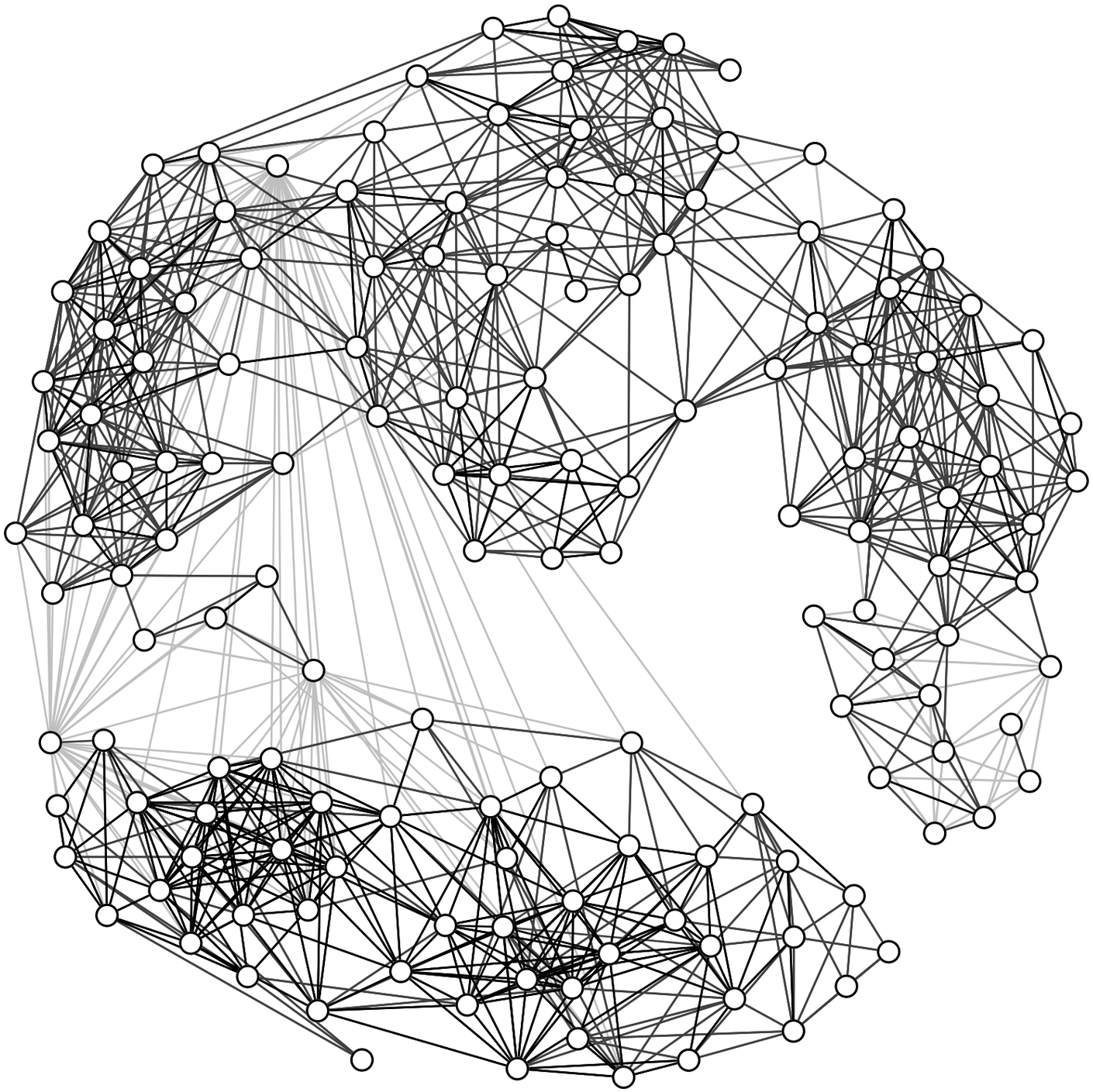
Getting non-complete graphs (2)

- How to decide when vertices are too dissimilar?
- In **ε-neighbour graphs** we add an edge between two vertices that are within distance ε 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 ε-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



ZM Fig. 16.1

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, $F = \{(v, u) \in E : |\{v, u\} \cap S| = 1\}$
- Graph cut clusters graph's vertices 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?

- Just any cut won't 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

What cuts would cut it? (1)

- The minimum cut usually just removes one vertex from the graph
  - 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 $W(A, B) = \sum_{i \in A, j \in B} w_{ij}$
    - w_{ij} is the weight of edge (i, j)

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

What cuts would cut it? (2)

- 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 not by $|C_i|$ but by $vol(C_i)$

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

Finding optimal RatioCut or NormalizedCut is NP-hard

Even More Matrices

- The (weighted) adjacency matrix $\boldsymbol{A}$ has the weight of edge (i, j) at position a_{ij}
- The **degree matrix** $\boldsymbol{\Delta}$ of a graph is a diagonal n-by-n matrix with the (weighted) degree of vertex i at position $\Delta_{ii} = d_i$
  - $\Delta_{ii} = d_i = \sum_j a_{ij}$
- 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
  - $\boldsymbol{M} = \boldsymbol{\Delta}^{-1}\boldsymbol{A}$

Graph Laplacians

- The **Laplacian matrix** $\boldsymbol{L}$ of a graph is the adjacency matrix subtracted from the degree matrix

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

- The Laplacian is symmetric and positive semi-definite
  - Undirected graphs
  - Has n real, non-negative, orthogonal eigenvalues
    $\lambda_1 \geq \lambda_2 \geq \lambda_3 \geq \ldots \geq \lambda_n \geq 0$

The normalized, symmetric Laplacian

- The **normalized, symmetric Laplacian matrix** $\boldsymbol{L}^s$ of a graph is defined as

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

  - Also positive semi-definite

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

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

$$= \boldsymbol{c}_i^T \boldsymbol{L} \boldsymbol{c}_i$$

Cuts using matrices

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

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

Finding approximate cuts

- Re-writing the objective functions doesn't make them any easier
  - But the complexity comes from the fact that we have to have binary clustering assignments
- Relax!
  - Let $\boldsymbol{c}_i$'s take any real value
- Relaxed RatioCut now looks like

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

  - $\boldsymbol{u}_i = \boldsymbol{c}_i / \|\boldsymbol{c}_i\|$ 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:

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

- Hence, $\boldsymbol{L}\boldsymbol{u}_i = \lambda_i \boldsymbol{u}_i$
  - $\boldsymbol{u}_i$ is an eigenvector of $\boldsymbol{L}$ corresponding to the eigenvalue λ_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
  - They are our relaxed cluster indicators
- Note that we know that $\lambda_n = 0$ and that the corresponding eigenvector is $(n^{-1/2}, n^{-1/2}, \ldots, n^{-1/2})$
  - No help on clustering...

Normalized cut and choice of Laplacians

- 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 normalized cut aim at minimizing intra-cluster similarity
  - But only normalized cut considers inter-cluster similarity
    $\Rightarrow$ Either $\boldsymbol{L}^s$ or $\boldsymbol{L}^a$
- The asymmetric Laplacian is better
  - With symmetric one further normalization is needed

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. if 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

Spectral clustering pseudo-code

Assume connected graph

Algorithm 16.1: Spectral Clustering Algorithm

SPECTRAL CLUSTERING $(\mathbf{D}, k)$:

1 Compute the similarity matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$

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

3 **else if** *normalized cut* **then** $\mathbf{B} \leftarrow \mathbf{L}^s$ or $\mathbf{L}^a$

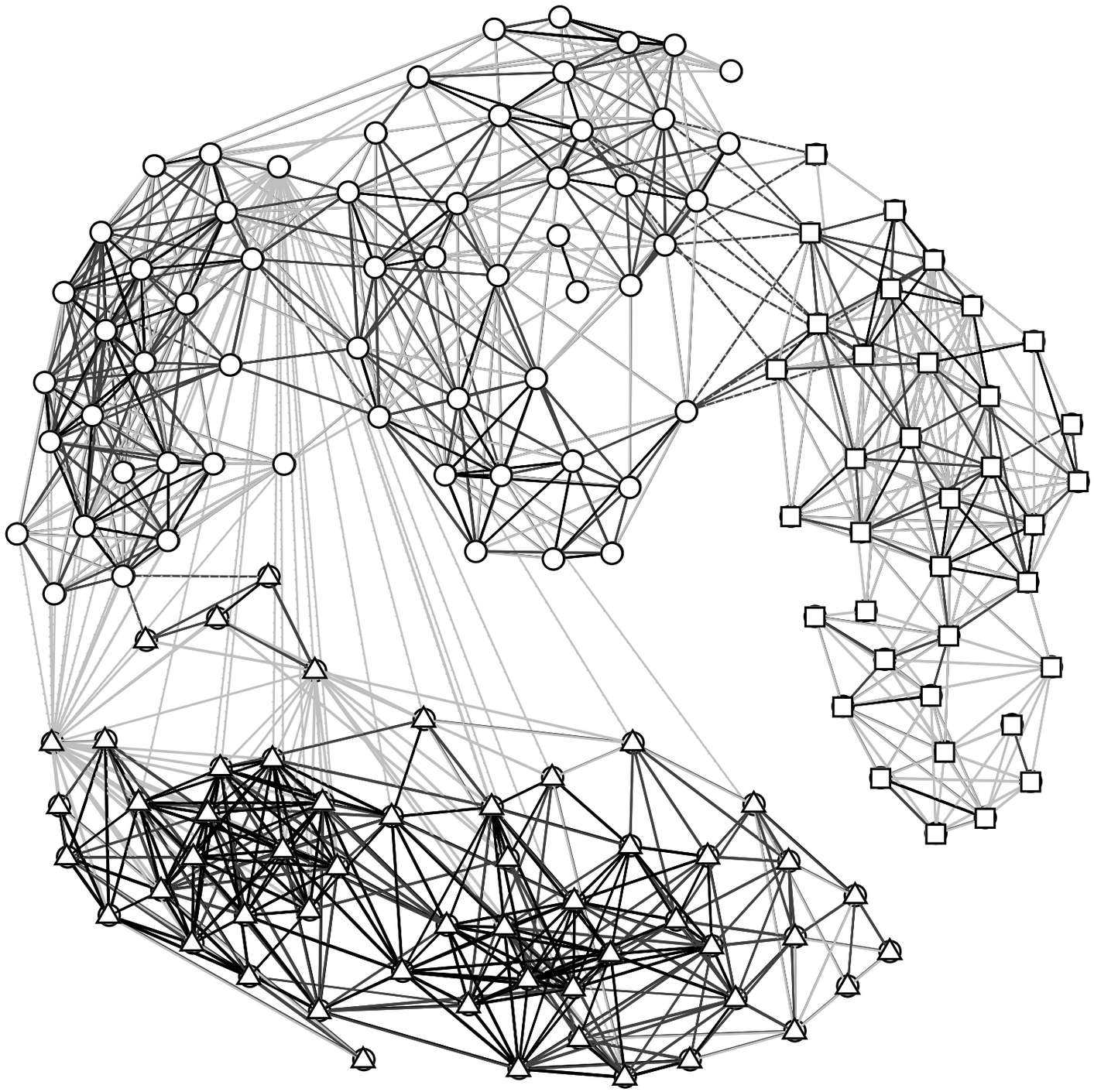
4 Solve $\mathbf{B}\mathbf{u}_i = \lambda_i \mathbf{u}_i$ for $i = n, \ldots, n-k+1$, where $\lambda_n \leq \lambda_{n-1} \leq \cdots \leq \lambda_{n-k+1}$

5 $\mathbf{U} \leftarrow \begin{pmatrix} \mathbf{u}_n & \mathbf{u}_{n-1} & \cdots & \mathbf{u}_{n-k+1} \end{pmatrix}$

6 $\mathbf{Y} \leftarrow$ normalize rows of $\mathbf{U}$ using (16.19)

7 $\mathcal{C} \leftarrow \{C_1, \ldots, C_k\}$ via K-means on $\mathbf{Y}$

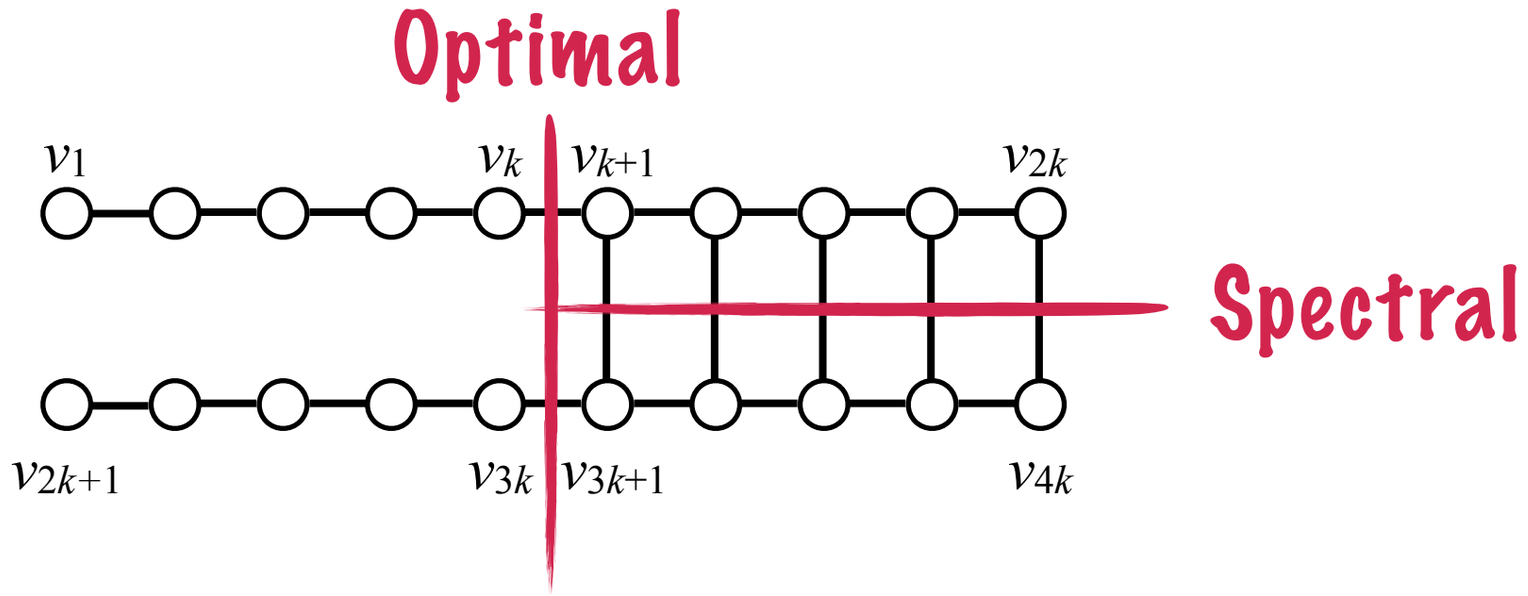
Example



ZM Figures 16.1 and 16.4

Is spectral clustering optimal?

- Spectral clustering is not always a good approximation of the graph cuts
  - In so-called cockroach graphs, spectral clustering always horizontally, when optimal is to cut vertically
  - Approximation ratio of $O(n)$



Markov Clustering

- A random walk on a graph that is in vertex v should visit other vertices from v's cluster more probably than vertices in other clusters
  - Transition probabilities are the edge weights, i.e. similarity counts
- Normalized adjacency matrix $\boldsymbol{M} = \boldsymbol{\Delta}^{-1}\boldsymbol{A}$ gives transition probabilities for a Markov chain
  - $\boldsymbol{M}^t = \boldsymbol{M} \times \boldsymbol{M} \times \ldots \times \boldsymbol{M}$ gives the probabilities to move from node i to node j in t steps
  - For i and j to be in the same cluster, these probabilities should be high versus what they are if i and j are in different clusters

Transition probability inflation

- The probabilities in $\boldsymbol{M}^t$ might not make the differences obvious enough
- We can inflate the probabilities by applying to every element of $\boldsymbol{M}$ the inflation operator

$$\Upsilon(\boldsymbol{M}, r) = \left(\frac{(m_{ij})^r}{\sum_{a=1}^{n} (m_{ia})^r} \right)_{ij}$$

  - This increases larger probabilities and reduces smaller ones

Markov clustering algorithm

- Compute $\boldsymbol{M}^1 \leftarrow \boldsymbol{\Delta}^{-1}\boldsymbol{A}$
  - Add self-edges to $\boldsymbol{A}$ if they don't exist
- **repeat**
  - $\boldsymbol{M}^t \leftarrow \boldsymbol{M}^{t-1} \times \boldsymbol{M}$
  - $\boldsymbol{M}^t \leftarrow \Upsilon(\boldsymbol{M}^t, r)$
- **until** successive $\boldsymbol{M}^t$'s don't change much
  - E.g. Frobenius is below a given threshold
- **return** clusters induced by $\boldsymbol{M}^t$

How to get the clusters

- $\boldsymbol{M}^t$ induces a weighted, directed graph G
  - Weight for edge (i, j) is the current transition probability from i to j
- In this graph, a vertex i is called **attractor** if it has a self-loop with positive probability
  - N.B. expects very small probabilities to be rounded to zero
- Attractor j **attracts** i if edge (i, j) has non-zero probability
- The number of clusters is the number of strongly connected components of attractors in G
  - These are the initial clusters
- Other vertices are attached to all clusters they can reach

Some notes

- The inflation parameter r implicitly defines the number of clusters
  - Higher $r \Rightarrow$ more clusters
- The convergence criterion can also have some effects
- Time complexity is $O(tn^{\omega})$
  - ω is the exponent for matrix multiplication
    - In practice $\omega = 3$ for full matrices and $\omega = 2$ for sparse matrices
    - Matrix $\boldsymbol{M}^t$ usually becomes sparse quickly

Summary

- Frequent subgraph mining can find 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
    - But this doesn't necessarily make it a good clustering algorithm
  - Markov clustering doesn't need the number of clusters
    - But does need the number of clusters