Statistical Data Analysis

Dr. Jana de Wiljes

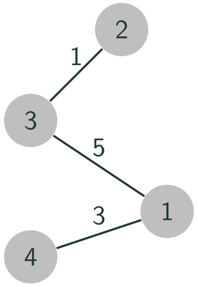
11. Januar 2022

Universität Potsdam

What is a graph (formally)?

The objects on the following slides will play a major role in this course.

- $G = (V, E, \omega)$, where $V \neq \emptyset$ is a set (called the **vertex set**), $E \subset \binom{V}{2} = \{\{u, v\} : u, v \in V\}$ (called the **edge set**) and $\omega : E \to \mathbb{R}^+$, is called a **(weighted) graph**
- usually we choose (or rename) $V = \{1, 2, \ldots, n\}$ and use the notations $ij = \{i, j\}$ for $\{i, j\} \in E$ and $\omega_{ij} = \omega(ij)$
- for every $i \in V$ define $N(i) := \{j \in V : ij \in E\}$, called the **neighbourhood** of i (in G); elements of $N(i)$ are called **neighbours** of i (those elements are **adjacent** to i)



$w(23) = 1$,
$N(4) = \{1\}$,
$d(1) = |\{3, 4\}| = 2$

Graph classes

Well known graph classes are:

- the **path graph** P_n has vertex set $\{1, 2, \ldots, n\}$ and edge set $\{\{1, 2\}, \{2, 3\}, \ldots, \{n-1, n\}\}$
- the **cycle graph** C_n has vertex set $\{1, 2, \ldots, n\}$ and edge set $\{\{1, 2\}, \{2, 3\}, \ldots, \{n-1, n\}, \{n, 1\}\}$
- the **complete graph** K_n consists of n vertices which are all adjacent to each other
- the **complete bipartite graph** $K_{m,n}$ has two sets V_1 and V_2 of vertices of sizes m and n, such that the edge set consists of all possible edges between V_1 and V_2

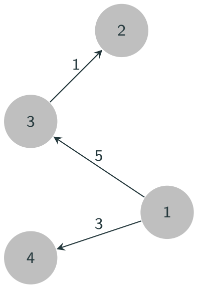
A set of vertices in a graph which are all adjacent to each other (they **induce** a complete (sub)graph), is called **clique**.

The graph $K_{1,n}$ is called a **star**.

What is a digraph (formally)?

Edges can have a direction.

- $G = (V, E, \omega)$, where $V \neq \emptyset$ is a set, $E \subset V \times V$ (this is sometimes also called the **set of arcs**) and $\omega : E \to \mathbb{R}^+$, is called a **(weighted) digraph**
- for $(i, j) \in E$ the vertex i is called **predecessor** of j and j is called **successor** of i
- similar notation simplifications as before
- $N^+(i) := \{j \in V : (i, j) \in E\}$ is the **out-neighbourhood** of i, $N^-(i) := \{j \in V : (j, i) \in E\}$ is the **in-neighbourhood** of i
- $d^+(i) := |N^+(i)|$ is the **out-degree** of i and $d^-(i) := |N^-(i)|$ is the **in-degree** of i



$N^-(3) = \{1\}$,
$N^+(4) = \emptyset$,
$d^+(1) = 2$,
$d^-(2) = 1$

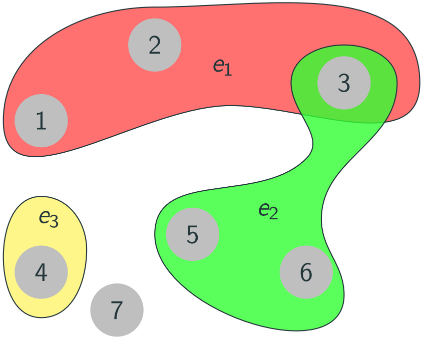
Example of a multigraph

It is sometimes necessary to allow multiple edges between two vertices or a **loop** (a self-edge). In that case we use the term **multigraph**.

What is a hypergraph (formally)?

Sometimes more than two vertices need to form an edge (certain real life situations' have this property).

- natural generalisation is a **hypergraph** $H = (V, E)$, where
  - $V \neq \emptyset$ is (also) a set, but
  - E can be an arbitrary subset (the elements are called **hyperedges**) of the power set $\mathcal{P}(V)$
- if all hyperedges are of the same size r, then H is called r-**uniform**



Storing graphs

Certain matrices and lists can be associated with a graph (we will see more examples later).

- **affinity matrix** $W(G)$:

$$w_{ij} = \begin{cases} \omega_{ij} & \text{if } \{i,j\} \in E, \\ 0 & \text{else.} \end{cases}$$

- **adjacency matrix** $A(G)$: special case of $W(G)$, where $w_{ij} = 1$ for all $ij \in E$.
- **adjacency list**:
  - associate list to every vertex containing its neighbours
  - call list of these lists adjacency list of the graph (treated differently in the literature)
  - not very useful for mathematical arguments
  - especially useful (for storing) when $A(G)$ is sparse

All the above constructions are valid for directed graphs.

How to transform a digraph into a graph?

Consider the following three approaches.

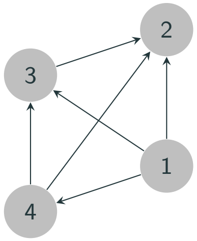
- ignore the directions
- carry out **cocitation coupling**
  - existence of common predecessors induce edges
  - weights are naturally given by number of common predecessors
- carry out **bibliographic coupling**
  - existence of common successors induce edges
  - weights are naturally given by number of common successors

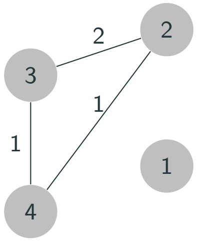
Cocitation coupling

A (undirected) graph is constructed via:

- **cocitation** c_{ij} of $i, j \in V$ is the number of common predecessors of i and j
- the **cocitation network** has vertex set V and an edge between i and j iff $c_{ij} > 0$
- it is also possible to obtain a weighted graph with weights c_{ij}
- note that $c_{ij} = \sum_{k=1}^{n} a_{ki} a_{kj}$, therefore $C = A^T A$

$$\longrightarrow$$



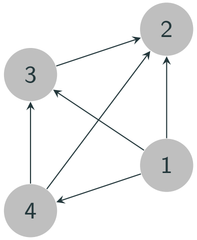


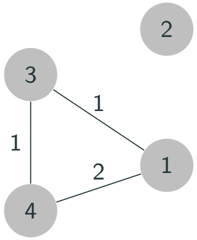
Bibliographic coupling

A (undirected) graph is constructed via:

- **bibliographic coupling** b_{ij} of $i, j \in V$ is the number of common successors of i and j
- the **bibliographic coupling network** has vertex set V and an edge between i and j iff $b_{ij} > 0$
- it is also possible to obtain a weighted graph with weights b_{ij}
- note that $b_{ij} = \sum_{k=1}^{n} a_{ik} a_{jk}$, therefore $B = AA^T$

$$\longrightarrow$$





How to transform a hypergraph into a graph?

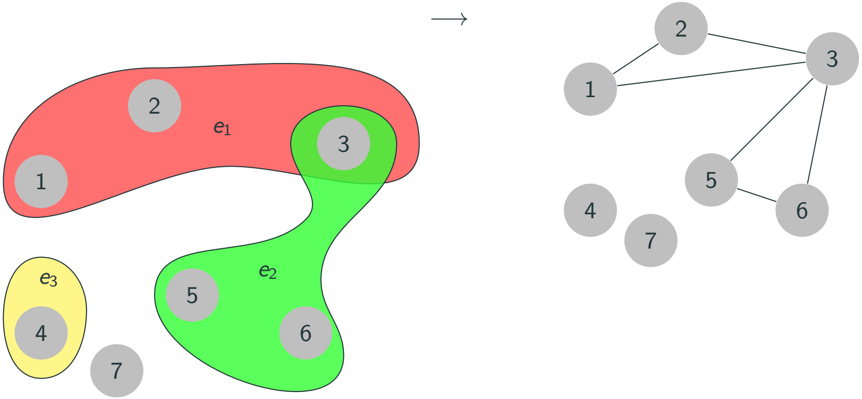
The following constructions are standard.

- clique expansion
  - the vertex set is V
  - each hyperedge e is replaced by an edge for every pair of vertices in e
  - this construction yields cliques for every hyperedge
- star expansion
  - vertex set is $V \cup E$
  - edge between u and e iff $u \in e$
  - every hyperedge corresponds to a star
- there are more. . .

Clique expansion

The clique expansion $G^{x} = (V^{x}, E^{x})$ is constructed from $H = (V, E)$ via:

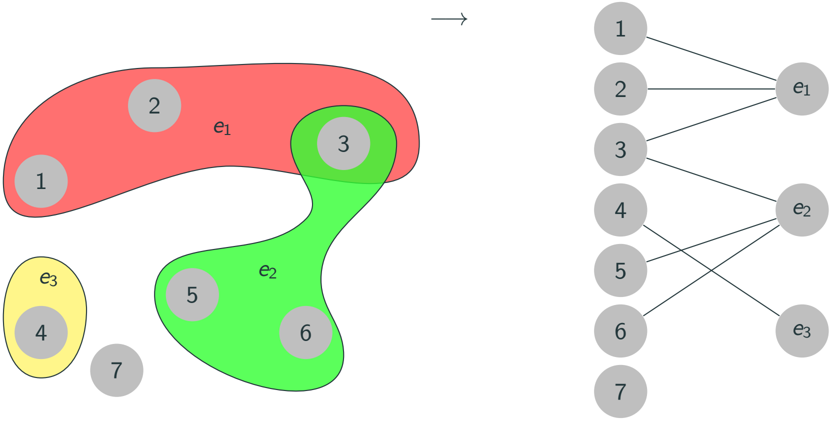
- $V^{x} = V$
- $E^{x} = \{\{i, j\} : \exists e \in E \text{ with } i, j \in e\}$



Star expansion

The star expansion $G^* = (V^*, E^*)$ is constructed from $H = (V, E)$ via:

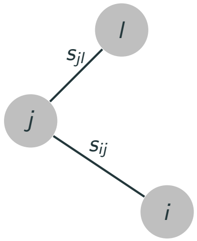
- $V^* = V \cup E$
- $E^* = \{\{i, e\} : i \in e, e \in E\}$



What if data without network structure is given?

Solution: Build your own graph!

- given a set of data points $x_1, x_2 \ldots, x_n$ and some notion of similarity[1] $s_{ij} \geq 0$ between all pairs of data points x_i and x_j
- build graph $G = (V, E)$, where the vertex i represents the data point x_i, so $V = \{1, 2, \ldots, n\}$
- $\{i, j\} \in E$ if $s_{ij} > 0$
- edge weight $\omega_{ij} = s_{ij}$ (edge weights represent similarities)
- G is called **similarity graph** (although with this particular choice of edges it is often referred to as the **fully connected graph**)



graph for $\{x_i, x_j, x_l\}$ with $s_{ij}, s_{jl} > 0$ and $s_{il} = 0$

The ε-neighbourhood graph

The ε-**neighbourhood graph** is constructed as follows:

- vertices are data points
- fix some $\varepsilon > 0$
- connect all vertices whose similarities are smaller than ε
- since ε is usually small, values of existing edges are roughly of the same scale
- hence usually unweighted

The (mutual) k-nearest neighbour graph

The k-**nearest neighbour graph** is constructed as follows:

- vertices are data points
- fix some $k > 0$
- connect i to the k nearest (w.r.t. s_{ij}) k vertices via an edge starting at i
- obtain an undirected graph by ignoring the directions

The **mutual** k-**nearest neighbour graph** is constructed as follows:

- vertices are data points
- fix some k
- connect i to the k nearest (w.r.t. s_{ij}) k vertices via an edge starting at i
- obtain an undirected graph by deleting all non symmetric edges

Graph Partitioning and Community Detection

Difference

Graph Partitioning (GP)

- partition vertices into given number of groups
- sizes of groups are (roughly) fixed
- many edges inside groups, few edges between groups
- goal: dividing network into smaller more manageable pieces
- example:
  - numerical solution of network processes on a parallel computer

Community Detection (CD)

- partition vertices into groups
- sizes of groups are not fixed
- many edges inside groups, few edges between groups
- goal: understanding structure of a network
- examples:
  - collaboration
  - related web pages

Why is partitioning hard?

Problem

Partition vertex set into two parts (*graph bisection*).

n vertices into parts of sizes n_1 and n_2 ($n_1 + n_2 = n$):

- $\frac{n!}{n_1! n_2!}$ possibilities (half of it if order is ignored and $n_1 = n_2$)
- using Stirling's formula $n! \approx \sqrt{2\pi n}(n/e)^n$ we get

$$\frac{n!}{n_1! n_2!} \approx \frac{n^{n+1/2}}{n_1^{n_1+1/2} n_2^{n_2+1/2}}$$

- for a balanced partition ($n_1 \approx n_2$):

$$\text{roughly } \frac{2^{n+1}}{\sqrt{n}} \text{ possibilities}$$

Therefore, exhausitive search is usually unfeasible.

Arbitrary number of classes

Methods for graph bisection can be generalised (other ways are possible):

- number of vertices: n
- number of classes: k
- define $l = k - 2^{\lfloor \log_2 k \rfloor}$
- for $r = 1, \ldots, l$ (ascending order) apply graph bisection method with

$$n_1^{(r)} = n - \frac{r \cdot n}{k} \quad \text{and} \quad n_2^{(r)} = \frac{n}{k}$$

- apply (equally sized) graph bisection $\lfloor \log_2 k \rfloor$ times to the $n - \frac{l \cdot n}{k}$ vertices in the large class
- results in k classes of (almost) same size $\frac{n}{k}$

Graph cuts

Given a graph $G = (V, E, \omega)$.

- for disjoint $A, B \subset V$ define the *cut size*

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

- for a partition $\mathcal{A} = A_1, A_2, \ldots, A_k$ define

$$\mathrm{cut}(A_1, A_2, \ldots, A_k) = \sum_{i=1}^{k} \mathrm{cut}(A_i, \overline{A}_i)$$

- basic problem: find $\mathcal{A}$ minimizing $\mathrm{cut}(A_1, A_2, \ldots, A_k)$

Types of methods (usually heuristics)

Local

- Kernighan-Lin algorithm
- Fiduccia-Mattheyses algorithm
- ...

Global

- Spectral clustering
- ...

Kernighan-Lin algorithm

The algorithm (Kernighan and Lin in 1970) works as follows:

Algorithmus (H)

Graph $G = (V, E)$*, positive integers* n_1, n_2 *Choose random partition* V_1, V_2 *of* V *with* $|V_l| = n_l$ *no pair of unused vertices exists minPair* $\leftarrow \sum \omega_{ij} + 1$ $i \in V_1, j \in V_2$*, neither* i *nor* j *has been swapped before*
$\mathrm{cut}\left((V_1 \setminus \{i\}) \cup \{j\}, (V_2 \setminus \{j\}) \cup \{i\}\right) \leq \mathrm{minPair}$ *minPair*
$\leftarrow \mathrm{cut}\left((V_1 \setminus \{i\}) \cup \{j\}, (V_2 \setminus \{j\}) \cup \{i\}\right)$ $i_0 \leftarrow i, j_0 \leftarrow j$
$V_1 = (V_1 \setminus \{i_0\}) \cup \{j_0\}, V_2 = (V_2 \setminus \{j_0\}) \cup \{i_0\}$ *(swap* i_0 *and* j_0*)*
Pick partition (from the different V_1, V_2*) with smallest cut size*
Partition V_1, V_2 *of* V *Partition vertex set into two parts (Kernighan & Lin)*

Runtime of Kernighan-Lin algorithm

At first glance we get:

- $0 \leq \min\{n_1, n_2\} \leq \frac{n}{2} \rightsquigarrow \mathcal{O}(n)$ swaps (worst case)
- $\frac{n}{2} \cdot \frac{n}{2}$ of considered pairs per swap (worst case)
- cut size change (swap i and j):

$$\sum_{v \in N(i) \cap V_1} \omega_{iv} + \sum_{v \in N(j) \cap V_2} \omega_{jv} - \sum_{v \in N(i) \cap V_2} \omega_{iv} - \sum_{v \in N(j) \cap V_1} \omega_{jv} + \omega_{ij}$$

- evaluating this expression costs $\mathcal{O}(m/n)$, where $m = |E|$, if stored in adjacency list
- $\rightsquigarrow$ runtime is $\mathcal{O}(m \cdot n^2)$ for one (!) round

Notes on the Kernighan-Lin algorithm

- swapping every round avoids local minima
- usually the algorithm is repeated several times with last output as input
- first partition is random $\rightsquigarrow$ sometimes algorithm is started more than once to get other (hopefully better) results
- convergence rate unknown
- for larger number of classes (i.e. $k \geq 3$) optimum might not be reached if applied
- Generalisation to hypergraphs: Fiduccia-Mattheyses algorithm

Spectral clustering

- mathematical foundation by Donath & Hoffman and Fiedler in 1973
- applications in various fields/for various problems
  - image segmentation
  - educational data mining
  - entity resolution
  - speech separation
  - . . .

Laplacian matrix (and another graph definition)

The degree matrix $D(G)$ is given by

$$d_{ij} = \begin{cases} \sum_{l \in N(i)} w_{il} & \text{if } i = j, \\ 0 & \text{else.} \end{cases}$$

Laplacian matrix:

$$L(G) = D(G) - W(G)$$

We also need:

$$\text{vol}(A) = \sum_{ij \in E, i,j \in A} \omega_{ij} \text{ for } A \subset V \text{ (no double counting!)}$$

Spectral clustering and graph cuts

Clustering corresponds to finding cuts in graphs (see next slides; other interpretations possible), there are (usually) two types:

- RatioCut (balanced by number of vertices in each cluster)
- NCut (balanced by sum of edge weights in each cluster)

Formalization of RatioCut and NCut

- $\text{RatioCut}(A_1, A_2, \ldots, A_k) = \sum_{i=1}^{k} \frac{\text{cut}(A_i, \overline{A}_i)}{|A_i|}$
    - $\sum_{i=1}^{k}(1/|A_i|)$ is minimized if all A_i have the same size
    - hence minimizing RadioCut balances clusters by their number of vertices (as desired)
- $\text{NCut}(A_1, A_2, \ldots, A_k) = \sum_{i=1}^{k} \frac{\text{cut}(A_i, \overline{A}_i)}{\text{vol}(A_i)}$
    - $\sum_{i=1}^{k}(1/\text{vol}(A_i))$ is minimized if all $\text{vol}(A_i)$ coincide
    - hence minimizing NCut balances clusters by their edge weights (as desired)
- corresponding optimization problems are NP hard
- spectral clustering is a way to solve relaxed versions of those problems

Useful properties of the Laplacian

Proposition: The Laplacian matrix L ($= D - W$) satisfies the following properties:

i For every vector $f \in \mathbb{R}^n$ we have

$$f^T L f = \frac{1}{2} \sum_{i,j=1}^{n} w_{ij} (f_i - f_j)^2.$$

ii L is symmetric and positive semi-definite.

iii The smallest eigenvalue of L is 0, the corresponding eigenvector is the constant one vector.

iv L has n non-negative, real-valued eigenvalues $0 = \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$.

Useful properties of the Laplacian

Proof:

i By the definition of d_i,

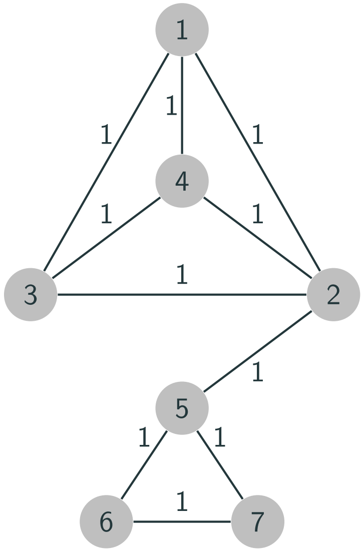
$$f^T L f = f^T D f - f^T W f = \sum_{i=1}^{n} d_i f_i^2 - \sum_{i,j=1}^{n} f_i f_j w_{ij}$$
$$= \frac{1}{2}\left(\sum_{i=1}^{n} d_i f_i^2 - 2\sum_{i,j=1}^{n} f_i f_j w_{ij} + \sum_{j=1}^{n} d_j f_j^2\right) = \frac{1}{2}\sum_{i,j=1}^{n} w_{ij}(f_i - f_j)^2.$$

ii The symmetry of L follows directly from the symmetry of W and D. The positive semi-definiteness is a direct consequence of Part (i), which shows that $f^T L f \geq 0$ for all $f \in \mathbb{R}^n$.

iii All eigenvalues are real (symmetric matrix). The rest follows easily from Part (ii) and the defining equation of eigenvalues.

iv This is a direct consequence of (i)-(iii).

Graph cuts and spectral clustering



$$f^T L f = \frac{1}{2} \sum_{i,j=1}^{n} w_{ij}(f_i - f_j)^2$$

Connectivity

Proposition: The number of connected components of a graph is equal to the multiplicity of the first eigenvalue (which is 0) of the graph Laplacian matrix.

Relaxation of RatioCut for $k = 2$

Goal is to solve

$$\min_{A \subset V} \text{RatioCut}(A, \overline{A})$$

Reformulation: for $A \subset V$ define $f = (f_1, \dots, f_n) \in \mathbb{R}^n$ (some kind of indicator vector) via

$$f_i = \begin{cases} \sqrt{|\overline{A}|/|A|} & \text{if } v_i \in A, \\ -\sqrt{|A|/|\overline{A}|} & \text{if } v_i \in \overline{A}. \end{cases}$$

It can be shown that

$$f^T L f = |V| \cdot \text{RatioCut}(A, \overline{A})$$

Relaxation of RatioCut for $k = 2$

Proof:

$$
\begin{aligned}
2f^T L f &= \sum_{i,j=1}^{n} w_{ij}(f_i - f_j)^2 \\
&= \sum_{i\in A, j\in\overline{A}} w_{i,j} \left(\sqrt{\frac{|\overline{A}|}{|A|}} \right. \\
&\quad \left. + \sqrt{\frac{|A|}{|\overline{A}|}} \right)^2 + \sum_{i\in\overline{A}, j\in A} w_{i,j} \left(-\sqrt{\frac{|\overline{A}|}{|A|}} - \sqrt{\frac{|A|}{|\overline{A}|}} \right)^2 \\
&= 2\mathrm{cut}(A, \overline{A}) \left(\frac{|\overline{A}|}{|A|} + \frac{|A|}{|\overline{A}|} + 2 \right) \\
&= 2\mathrm{cut}(A, \overline{A}) \left(\frac{|A| + |\overline{A}|}{|A|} + \frac{|A| + |\overline{A}|}{|\overline{A}|} \right) \\
&= 2|V| \cdot \mathrm{RatioCut}(A, \overline{A})
\end{aligned}
$$

Relaxation of RatioCut for $k = 2$

Further, f is orthogonal to 1:

$$\sum_{i=1}^{n} f_i = \sum_{i \in A} \sqrt{\frac{|\overline{A}|}{|A|}} - \sum_{i \in \overline{A}} \sqrt{\frac{|A|}{|\overline{A}|}} = |A| \sqrt{\frac{|\overline{A}|}{|A|}} - |\overline{A}| \sqrt{\frac{|A|}{|\overline{A}|}} = 0.$$

Finally

$$\|f\|^2 = \sum_{i=1}^{n} f_i^2 = |A| \frac{|\overline{A}|}{|A|} + |\overline{A}| \frac{|A|}{|\overline{A}|} = |\overline{A}| + |A| = n.$$

Hence the minimization problem is equivalent to

$$\min_{A \subset V} f^T L f$$

subject to the given f.

Relaxation of RatioCut for $k = 2$

This problem is NP-hard since the solution vector only takes two particular values.

Relaxing this problem is possible:

$$\min_{f \in \mathbb{R}^n} f^T L f \text{ subject to } f \perp 1, \|f\| = \sqrt{n}$$

The Rayleigh-Ritz theorem (see e.g. Strang - Linear algebra and its applications) gives the solution of this problem via an eigenvector corresponding to the second smallest eigenvalue of L.

Remark: Solution has to be transformed!

Special case ($k = 2$) of unnormalized spectral clustering

Algorithmus (H)

Weight matrix (or any similarity matrix) $S \in \mathbb{R}^{n \times n}$ Construct similarity graph G Compute $L(G)$ Compute Eigenvectors X_1 and X_2 of $L(G)$ Build $U \in \mathbb{R}^{n \times 2}$ with X_1 and X_2 as columns Rows of U are $y_1, y_2, \ldots, y_n \in \mathbb{R}^2$ Cluster $y_1, y_2, \ldots, y_n$ into Clusters C_1 and C_2 (k-means) Clusters $A_1 = \{j : y_j \in C_1\}$ and $A_2 = \{j : y_j \in C_2\}$ Unnormalized spectral clustering ($k = 2$)

Relaxation of RatioCut for $k > 2$ (sketch)

- Define k indicator vectors $h_j = (h_{1,j}, h_{2,j}, \dots, h_{n,j})^T$ via

$$h_{i,j} = \begin{cases} 1/\sqrt{|A_j|} & \text{if } v_i \in A_j, \\ 0 & \text{else} \end{cases}$$

- Matrix $H \in \mathbb{R}^{n \times k}$ with columns $h_1, h_2, \dots, h_k$ is orthogonal
- We have $h_i^T L h_i = \frac{\text{cut}(A_i, \overline{A}_i)}{|A_i|}$ and $h_i^T L h_i = (H^T L H)_{ii}$
- Together this yields $\text{RatioCut}(A_1, A_2, \dots, A_k) = \text{tr}(H^T L H)$
- Hence the minimum of $\text{RatioCut}(A_1, A_2, \dots, A_k)$ can be approximated by solving (the relaxed version)

$$\min_{H \in \mathbb{R}^{n \times k}} \text{tr}(H^T L H) \text{ subject to } H^T H = I$$

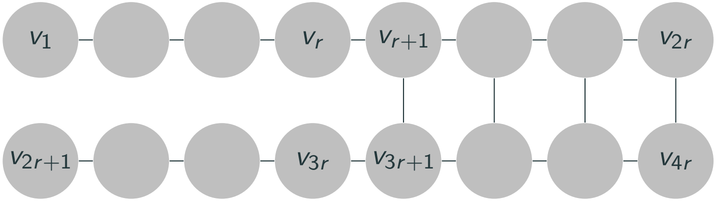
General case of unnormalized spectral clustering

Algorithmus (H)

Weight matrix (or any similarity matrix) $S \in \mathbb{R}^{n \times n}$, *number k of clusters Construct similarity graph G Compute* $L(G)$ *Compute Eigenvectors* $X_1, X_2, \ldots, X_k$ *of* $L(G)$ *Build* $U \in \mathbb{R}^{n \times k}$ *with* $X_1, X_2, \ldots, X_k$ *as columns Rows of U are* $y_1, y_2, \ldots, y_n \in \mathbb{R}^k$ *Cluster* $y_1, y_2, \ldots, y_n$ *into Clusters* $C_1, C_2, \ldots, C_k$ *(k-means) Clusters* $A_1, A_2, \ldots, A_k$ *with* $A_i = \{j : y_j \in C_i\}$ *Unnormalized spectral clustering*

Notes on spectral clustering

- faster ($\mathcal{O}(mn)$ or $\mathcal{O}(n^2)$ for the eigenvector calculation) than Kernighan-Lin
- quality of approximated solution can be arbitrarily far from exact solution (cockroach graphs for $k = 2$)



- other relaxations possible (and probably useful)
- any other clustering algorithm may be used instead of k-means

Computing eigenvalues

- find roots of characteristic polynomial (computationally expensive)
- power method: iterate (with any starting vector $X_{(0)}$)

$$X_{(l)} = A^l X_{(0)}$$

- power method converges to eigenvector X corresponding to (w.r.t. absolute value) largest eigenvalue
- power method is fast but
    - method does not work if $X_{(0)}$ is orthogonal to X (can be avoided by choosing all entries of $X_{(0)}$ to be positive, since X has only entries of same sign)
    - entries of $X_{(l)}$ become large during the iterative process (renormalization helps)
    - When are we done? (option is to start with two different vectors)

Computational complexity of the power method

Two aspects:

- complexity of one multiplication
- required number of multiplications

First aspect:

- n^2 multiplications if stored in adjacency matrix
- less if matrix is stored in adjacency lists and matrix is sparse
  - compute
    $$\sum_{j \in N(i)} X_{(l)_j}$$
    which gives the i-th entry of $X_{(l+1)}$
  - therefore a total of
    $$\sum_i d(i) = 2m$$
    operations (m being the number of edges in the graph)

Computational complexity of the power method

Second aspect:

- it can be shown that this is $\mathcal{O}(n)$
- we will come back to this

Total computational complexity is $\mathcal{O}(mn)$, which means

- $\mathcal{O}(n^2)$ if graph is sparse
- $\mathcal{O}(n^3)$ if graph is dense

$\rightsquigarrow$ use adjacency list

Computing other eigenvalues

We have

$$(\lambda_n I - L)X_i = (\lambda_n - \lambda_i)X_i$$

hence eigenvalues are reversed for matrix $\lambda_n I - L$

$\rightsquigarrow$ smallest eigenvalue can be calculated with power method

Computing other eigenvalues

Trick to compute second largest eigenvalue:

- X_n normalised eigenvector corresponding to largest eigenvalue λ_n
- choose starting vector X and define

$$Y = X - (X_n^T X) X_n$$

- we have

$$X_i^T Y = \begin{cases} 0 & \text{if } i = n, \\ X_i^T X & \text{otherwise} \end{cases}$$

- therefore

$$Y = \sum_{i=1}^{n-1} c_i X_i$$

  where $c_i = X_i^T Y$

$\rightsquigarrow$ use Y as starting vector for power method

Compute all eigenvalues and eigenvectors

Combining methods for given (symmetric) matrix A with eigenvectors X_i and eigenvalues λ_i:

- Find orthogonal matrix Q with $B = Q^T A Q$ being a tridiagonal matrix
    - $Q^T X_i$ is eigenvector of B
    - B can be found efficiently, e.g. by Householder algorithm or Lancosz algorithm
- compute eigenvalues and eigenvectors of B
    - these give eigenvalues and eigenvectors of A
    - can be done using for example the QL algorithm