Complex Networks
matrix algorithms and graph partitioning

2019.1.10(Thu)

contents of this chapter

- eigenvector centrality
- finding the leading eigenvector of the adjacency matrix
- spectral partitioning method
- network community detection

leading eigenvectors and eigenvector centrality

- eigenvector centrality of vertex i
  - ith element of the leading eigenvector of the adjacency matrix
- calculating the complete set of eigenvectors of the adjacency matrix is a wasteful approach
- power method: $\mathbf{x}(t) = \mathbf{A}^t \mathbf{x}(0)$
  - x(t) will converge to the leading eigenvector as $t \to \infty$
  - no faster method for calculating eigenvector centrality

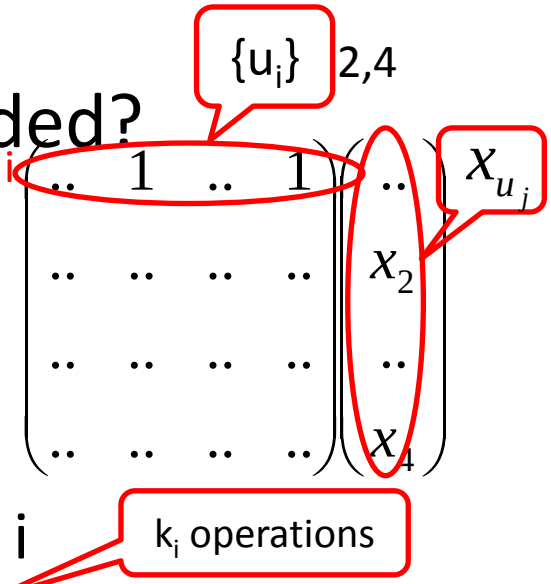
caveats of power method

- the method will not work if the initial vector x(0) happens to be orthogonal to the leading eigenvector
  - one simple way to avoid this problem is to choose the initial vector to have all elements positive
- the elements of the vector have a tendency to grow on each iteration
  - we must periodically renormalize the vector by dividing all the elements by the same value
- how long do we need to go on multiplying by the adjacency matrix before the result converges to the leading eigenvalue?
  - this will depend on how accurate an answer we require.

computational complexity of power method

1. how long each multiplication by the adjacency matrix takes?
2. how many multiplications are needed?

1.



- adjacency matrix : n^2 multiplication
- adjacency list : O(m)
  - $\{u_j\}, j = 1 \ldots k_i$:set of neighbors of vertex i
  - $[\mathbf{Ax}]_i$:ith element of $\mathbf{Ax}$ $\quad [\mathbf{Ax}]_i = \sum_{j=1}^{k_i} x_{u_j}$
  - all elements can be completed in time proportional to $\sum_i k_i = 2m$

how many multiplications must we perform?

2.

$\mathbf{x}(0) = \sum_i c_i \mathbf{v}_i$

constant (depends on the choice of initial vector)

normalized ith eigenvector

corresponding eigenvalue

$$\mathbf{x}(t) = \mathbf{A}^t \sum_i c_i \mathbf{v}_i = \sum_i c_i \kappa_i^t \mathbf{v}_i = \kappa_1^t \sum_i c_i \left[\frac{\kappa_i}{\kappa_1}\right]^t \mathbf{v}_i$$

$$\frac{\mathbf{x}(t)}{c_1 \kappa_1^t} = \mathbf{v}_1 + \frac{c_2}{c_1}\left(\frac{\kappa_2}{\kappa_1}\right)^t \mathbf{v}_2 + \ldots,$$

$$\sqrt{\left|\frac{\mathbf{x}(t)}{c_1 \kappa_1^t} - \mathbf{v}_1\right|^2} = \frac{c_2}{c_1}\left(\frac{\kappa_2}{\kappa_1}\right)^t$$

- if we want this error to be at most ε

$$t \geq \frac{\ln(1/\varepsilon) + \ln(c_1/c_2)}{\ln(\kappa_1/\kappa_2)}$$

maximum eigenvalue κ_1

minimum eigenvalue $\kappa_n \geq -|\kappa_1|$

} mean spacing $2\kappa_1/(n-1)$

$$\kappa_2 \approx \kappa_1 - a\,\kappa_1/n$$

$$\ln\frac{\kappa_1}{\kappa_2} \approx -\ln\left(1 - \frac{a}{n}\right) = \frac{a}{n} + O(n^{-2}) \Rightarrow t = O(n)$$

computational complexity of power method

- 1. each multiplication : O(m)
- 2. number of multiplication : O(n)

  } O(mn) time overall

  - sparse network ($m \propto n$) : $O(mn) \equiv O(n^2)$
  - dense network ($m \propto n^2$): $O(mn) \equiv O(n^3)$
- if adjacency matrix is used, multiplications take $O(n^2)$, so the total calculation takes $O(n^3)$

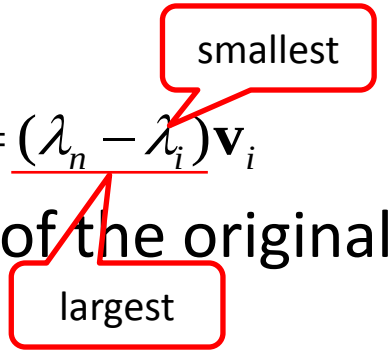
calculating other eigenvalues and eigenvectors

- power method calculates the largest eigenvalue of a matrix (and the corresponding eigenvector)
- method for finding the smallest eivenvalue
  - shifting all the eigenvalues by a constant amount
- eigenvalues of graph Laplacian L are non-negative

$$\lambda_1 \le \lambda_2 \le \ldots \le \lambda_n \quad \mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n$$

$$(\lambda_n \mathbf{I} - \mathbf{L})\mathbf{v}_i = (\lambda_n - \lambda_i)\mathbf{v}_i$$

  - $\mathbf{v}_i$ is an eivenvector of $(\lambda_n \mathbf{I} - \mathbf{L})\mathbf{v}_i = (\lambda_n - \lambda_i)\mathbf{v}_i$
  - their order is reversed from those of the original Laplacian
  - smallest eigenvalue of $\mathbf{L}$ ⇦ largest eigenvalue of $\lambda_n \mathbf{I} - \mathbf{L}$



finding the second-largest eigenvalue (1)

- $\mathbf{v}_1$: normalized eigenvector corresponding to the largest eigenvalue of a matrix $\mathbf{A}$
- define $\mathbf{y} = \mathbf{x} - (\mathbf{v}_1^T \mathbf{x})\mathbf{v}_1$
- this vector has the property that

$$\mathbf{v}_i^T \mathbf{y} = \mathbf{v}_i^T \mathbf{x} - (\mathbf{v}_1^T \mathbf{x})(\mathbf{v}_i^T \mathbf{v}_1) = \mathbf{v}_i^T \mathbf{x} - \mathbf{v}_1^T \mathbf{x}\delta_{i1} = \begin{cases} 0 & \text{if i = 1} \\ \mathbf{v}_i^T \mathbf{x} & \text{otherwise} \end{cases}$$

- y is equal to x along the direction of every eigenvector of A except the leading eigenvector
- $\mathbf{y} = \sum_{i=1}^{n} c_i \mathbf{v}_i$ with $c_i = \mathbf{v}_i^T \mathbf{y}$ has no term in $\mathbf{v}_1$ since $c_1 = \mathbf{v}_1^T \mathbf{y} = 0$

$$\therefore \mathbf{y} = \sum_{i=2}^{n} c_i \mathbf{v}_i$$

finding the second-largest eigenvalue (2)

- use vector y as the starting vector for repeated multiplication by A
- after multiplying y by A t times

$$\mathbf{y}(t) = \mathbf{A}^t \mathbf{y}(0) = \kappa_2^t \sum_{i=2}^{n} c_i \left[\frac{\kappa_i}{\kappa_2} \right]^t \mathbf{v}_i$$

less than 1 for all i > 2

- as $t \to \infty$, $y(t) \to \kappa_2^t c_2 v_2$
- caveats
  - y might have a very small component in the direction of v_1
    - periodically perform a subtraction

Gram-Schmidt orthogonalization

$$\mathbf{y}(t) = \mathbf{y}(t) - (\mathbf{v}_1^T \mathbf{y}(t))\mathbf{v}_1$$

- not work well beyond the first couple of eigenvectors

efficient algorithms for computing all eigenvalues and eigenvectors

- finding orthogonal matrix Q such that the similarity transform $\mathbf{T} = \mathbf{Q}^T \mathbf{A} \mathbf{Q}$ gives either
  - a tridiagonal matrix (if A is symmetric) or
  - a Hessenberg matrix (if A is asymmetric)
- if we can find such Q, and if $\mathbf{v}_i$ is an eigenvector of $\mathbf{A}$ with eigenvalue κ_i

$$\kappa_i \mathbf{Q}^T \mathbf{v}_i = \mathbf{Q}^T \mathbf{A} \mathbf{v}_i = \mathbf{T} \mathbf{Q}^T \mathbf{v}_i \qquad \because \mathbf{Q}^{-1} = \mathbf{Q}^T$$

Q is an orthogonal matrix

- the vector $\mathbf{w}_i = \mathbf{Q}^T \mathbf{v}_i$ is an eigenvector of T with eigenvalue κ_i
- eigenvectors of A are $\mathbf{v}_i = \mathbf{Q} \mathbf{w}_i$

QL algorithm: efficient numerical method
O(n) for a tridiagonal matrix
$O(n^2)$ for a Heessenberg one

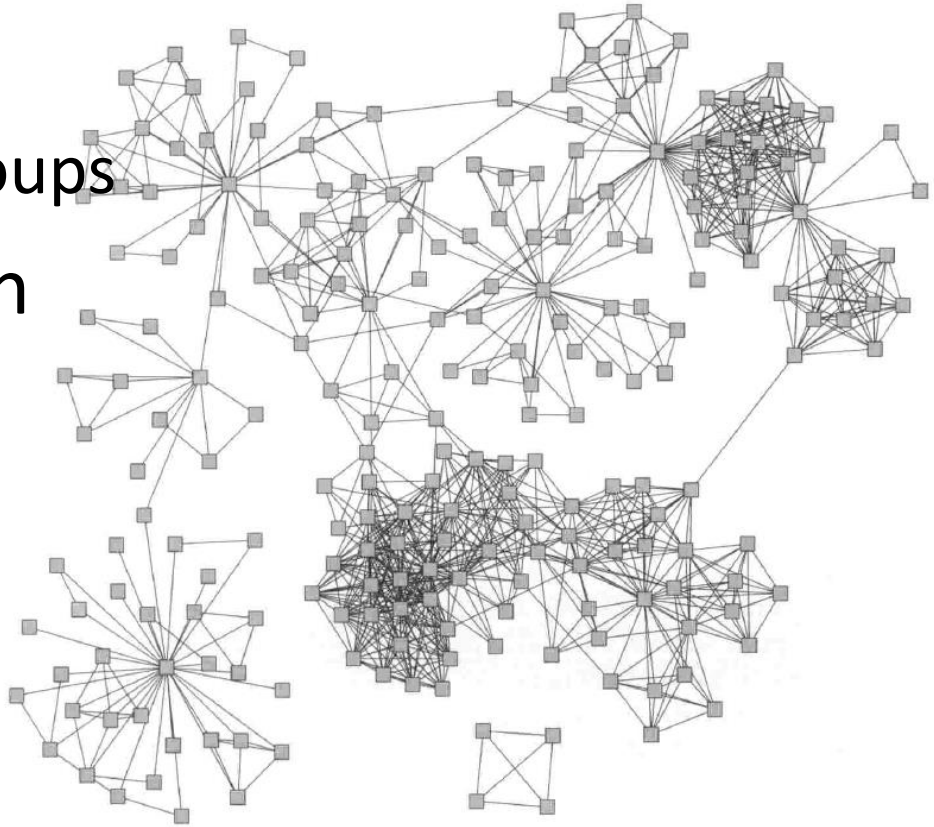
algorithms for finding Q

- for a symmetric matrix
  - Householder algorithm : $O(n^3)$
- for a sparse symmetric matrix
  - Lanczos algorithm : O(mn)
- for an asymmetric matrix
  - Arnoldi algorithm

- combined Lanczos/QL algorithm : O(mn)

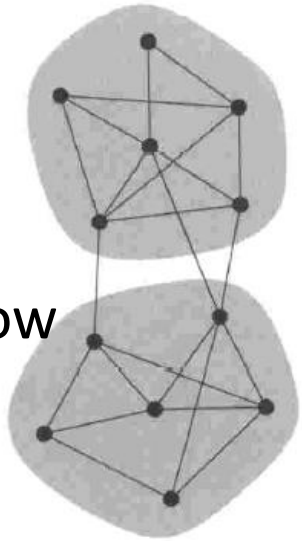
dividing networks into clusters

- “graph partitioning”, “community detection”
- network of coauthorships in a university department
  - densely connected groups
- discovering groups can be a useful tool for revealing structure and organization



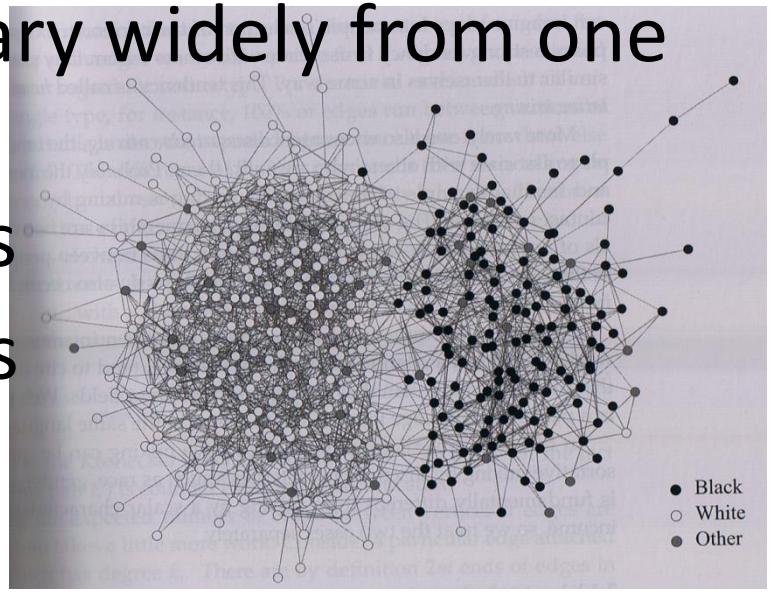
graph partitioning

- the number and size of the groups is fixed
  - dividing a network into two groups of equal size, such that the number of edges between them is minimized
  - example: network processes on a parallel computer
    - message transmission between processors is slow



community detection

- the number and size of the group is unspecified
  - finding the natural divisions of a network into groups of vertices such that there are many edges within groups and few edges between groups
- sizes of the groups might vary widely from one group to another
- a wide variety of definitions
- a wide variety of algorithms



difference between graph partitioning and community detection

- graph partitioning
  - the number and size of the groups is specified
  - dividing a network into smaller manageable pieces
- community detection
  - the number and size of the groups is unspecified
  - used as a tool for understanding the structure of a network

why partitioning is hard? (1)

- graph bisection : division into two parts
  - repeated bisection : division into arbitrary number of parts
- cut size : the number of edges between groups
- exhaustive search is prohibitively costly
  - the number of ways of dividing a network of n vertices into two groups of n_1 and n_2 vertices

$$\frac{n!}{n_1!n_2!} \cong \frac{\sqrt{2\pi n}(n/e)^n}{\sqrt{2\pi n_1}(n_1/e)^{n_1}\sqrt{2\pi n_2}(n_2/e)^{n_2}} = \frac{n^{n+1/2}}{n_1^{n_1+1/2}n_2^{n_2+1/2}} \quad \because n! \cong \sqrt{2\pi n}(n/e)^n, \; n = n_1 + n_2$$

why partitioning is hard? (2)

- if we want to divide a network into two parts of equal size $\frac{1}{2}n$, the number of different ways to do it is roughly

$$\frac{n^{n+1/2}}{(n/2)^{n+1}} = \frac{2^{n+1}}{\sqrt{n}}$$

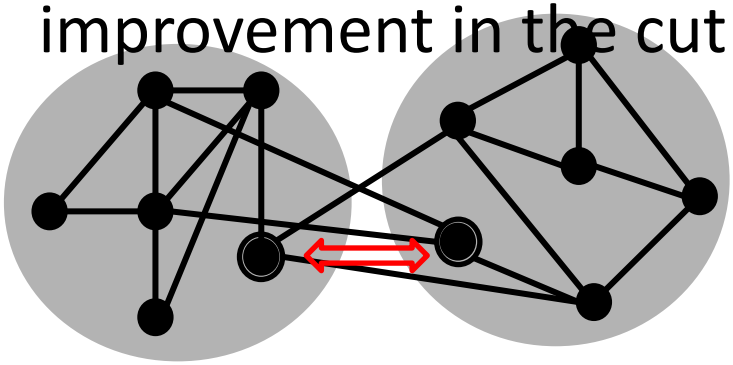
go up roughly exponentially with the size of the network

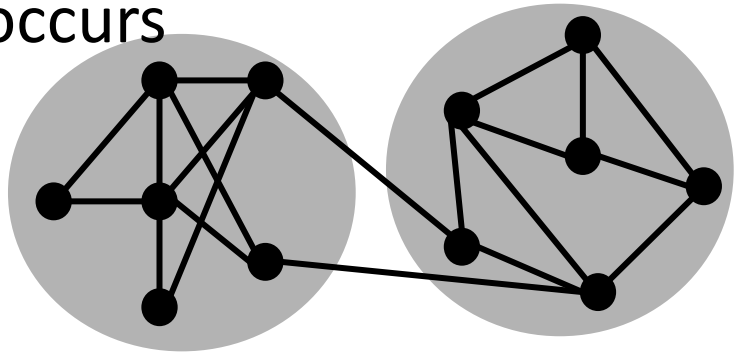
calculation becomes intractable for the networks beyond n=30

- Even algorithms that fail to find the very best division of a network may still find a pretty good one, and for many practical purposes pretty good is good enough.
  - heuristic algorithms

The Kernighan-Lin algorithm

1. start with any division of the vertices into two groups
2. search for pairs of vertices whose interchange would reduce the cut size between the groups by the largest amount (or increase it by the smallest amount)
3. repeat the process (each vertex can only be moved once)
4. when all swaps have been completed, go back through every state and choose the state in which the cut size takes its smallest value
5. this entire procedure is performed repeatedly until no improvement in the cut size occurs





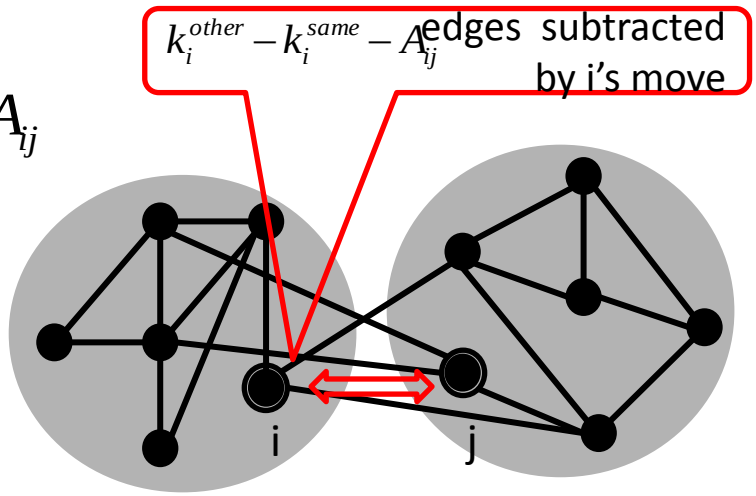
comments on the Kernighan-Lin algorithm

- if initial partitions are different, the results can be different
- the Kernighan-Lin algorithm is slow
  - # of swaps : O(n) — n/2
  - for each swap, examine all pairs of vertices : $O(n^2)$ — $(n/2) \times (n/2) = n^2/4$
  - check the changes after swap
    - the change of cut size

      $$\Delta = k_i^{other} - k_i^{same} + k_j^{other} - k_j^{same} - 2A_{ij}$$

    - O(m/n) — average degree
  - total : $O(n \times n^2 \times m/n) = O(mn^2)$

    $O(n^3)$ on a sparse network
    $O(n^4)$ on a dense network



spectral partitioning (1)

- a network of n vertices and m edges
- a division into group 1 and group 2
- cut size (the number of edges running between the two groups) $R = \sum_{\substack{i,j \text{ in} \\ \text{different} \\ \text{groups}}} A_{ij}$

$$s_i = \begin{cases} +1 & \text{if vertex i belongs to group 1} \\ -1 & \text{if vertex i belongs to group 2} \end{cases}$$

$$\frac{1}{2}(1 - s_i s_j) = \begin{cases} 1 & \text{if i and j are in different groups} \\ 0 & \text{if i and j are in the same group} \end{cases}$$

$$R = \frac{1}{4} \sum_{ij} A_{ij} (1 - s_i s_j)$$

spectral partitioning (2)

- the first term in the sum

$$\because \sum_j A_{ij} = k_i \qquad s_i^2 = 1$$

$$\sum_{ij} A_{ij} = \sum_i k_i = \sum_i k_i s_i^2 = \sum_{ij} k_i \delta_{ij} s_i s_j$$

$$R = \frac{1}{4}\sum_{ij}(k_i\delta_{ij} - A_{ij})s_i s_j = \frac{1}{4}\sum_{ij} L_{ij} s_i s_j$$

ijth element of graph Laplacian

$$R = \frac{1}{4}\mathbf{s}^T \mathbf{L}\mathbf{s}$$

division into groups

graph structure

- goal: find the vector s that minimizes the cut size for given L

spectral partitioning (3)

- minimizing R is not easy
  - s_i cannot take any values (+1 or -1)
- approximate approach: allow s_i to take any values
  - constraint1 : $|s| = \sqrt{n}$ or $\sum_i s_i^2 = n$
  - constraint2 : $\sum_i s_i = n_1 - n_2$ or $\mathbf{1}^T \mathbf{s} = n_1 - n_2$

if no constraint, **s**=0 is trivial solution

the length of vector **s** should not be changed

the sizes of two groups should be n_1 and n_2

spectral partitioning (4)

- differentiate with respect to the elements s_i
  - Lagrange multipliers

constraint1

constraint2

$$\frac{\partial}{\partial s_i}\left[\sum_{jk} L_{jk} s_j s_k + \lambda\left(n - \sum_i s_j^2\right) + 2\mu\left((n_1 - n_2) - \sum_j s_j\right)\right] = 0$$

$$\sum_j L_{ij} s_j = \lambda s_i + \mu$$

$$\mathbf{Ls} = \lambda \mathbf{s} + \mu \mathbf{1}$$

Laplacian matrix (L=D-A) is symmetric $(L\cdot 1)^T = 1^T L^T = 1^T L = 0$

- multiplying on the left by 1^T

$$\mu = -\frac{n_1 - n_2}{n}\lambda \qquad \because \mathbf{L}\cdot\mathbf{1} = 0 \qquad \mathbf{1}^T\mathbf{s} = n_1 - n_2$$

$$\mathbf{x} = \mathbf{s} + \frac{\mu}{\lambda}\mathbf{1} = \mathbf{s} - \frac{n_1 - n_2}{n}\mathbf{1}$$

define new vector x

$$\mathbf{Lx} = \mathbf{L}(\mathbf{s} + \frac{\mu}{\lambda}\mathbf{1}) = \mathbf{Ls} = \lambda\mathbf{s} + \mu\mathbf{1} = \lambda\mathbf{x}$$

x is an eigenvector of the Laplacian with eigenvalue λ

spectral partitioning (5)

- which eigenvector should we choose?

$$\mathbf{1}^T\mathbf{x} = \mathbf{1}^T\mathbf{s} + \frac{\mu}{\lambda}\mathbf{1}^T\mathbf{1} = (n_1 - n_2) - \frac{n_1 - n_2}{n}n = 0$$

- **x** is orthogonal to **1** -> **x ≠ 1**

$$\frac{1}{4}\mathbf{s}^T\mathbf{L}\mathbf{s} = \frac{1}{4}(\mathbf{x} - \frac{\mu}{\lambda}\mathbf{1})^T\mathbf{L}(\mathbf{x} - \frac{\mu}{\lambda}\mathbf{1})$$
$$= \frac{1}{4}(\mathbf{x}^T - \frac{\mu}{\lambda}\mathbf{1}^T)\mathbf{L}(\mathbf{x} - \frac{\mu}{\lambda}\mathbf{1})$$
$$= \frac{1}{4}\mathbf{x}^T\mathbf{L}(\mathbf{x} - \frac{\mu}{\lambda}\mathbf{1}) = \frac{1}{4}\mathbf{x}^T\mathbf{L}\mathbf{x}$$

$$R = \frac{1}{4}\mathbf{s}^T\mathbf{L}\mathbf{s} = \frac{1}{4}\mathbf{x}^T\mathbf{L}\mathbf{x} = \frac{1}{4}\lambda\mathbf{x}^T\mathbf{x}$$

$$\mathbf{x}^T\mathbf{x} = \mathbf{s}^T\mathbf{s} + \frac{\mu}{\lambda}(\mathbf{s}^T 1 + \mathbf{1}^T\mathbf{s}) + \frac{\mu^2}{\lambda^2}\mathbf{1}^T\mathbf{1}$$
$$= n - 2\frac{n_1 - n_2}{n}(n_1 - n_2) + \frac{(n_1 - n_2)^2}{n}n = 4\frac{n_1 n_2}{n} \qquad R = \frac{n_1 n_2}{n}\lambda$$

- cut size is proportional to the eigenvalue λ
  - smallest eigenvalue is 0 that corresponds to eigenvector **1**
  - choose **x** proportional to the eigenvector v_2 corresponding to the second smallest eigenvalue λ_2

but **x≠1**

$$\mathbf{s} = \mathbf{x} + \frac{n_1 - n_2}{n}\mathbf{1}$$

s_i should be +1 or -1

$$\mathbf{s}^T\left(\mathbf{x} + \frac{n_1 - n_2}{n}\mathbf{1}\right) = \sum_i s_i\left(x_i + \frac{n_1 - n_2}{n}\right)$$

make this product as large as possible

algorithm of spectral partitioning

1. calculate the eigenvector v_2 corresponding the second smallest eigenvalue λ_2 of the graph Laplacian
2. sort the elements of the eigenvector in order from largest to smallest
3. put the vertices corresponding to the n_1 largest elements in group 1, the rest in group 2, and calculate the cut size
4. then put the vertices corresponding to the n_1 smallest elements in group 1, the rest in group 2, and recalculate the cut size
5. between these two divisions of the network, choose the one that gives the smaller cut size

comments on spectral partitioning

- disadvantages
  - quality of partition: not quite as good as those returned by other methods
- advantages
  - speed :
    - calculation of the eigenvector v_2 : O(mn)

      (or $O(n^2)$ for sparse networks)

Kernighan-Lin algorithm: $O(n^3)$

community detection

- difference from graph partitioning : the number or size of the groups is not fixed
- if no constraint, optimum division is trivial
  - group 1: all vertices
  - group 2: no vertex

  } cut size = 0
- loose constraint
  - minimize ratio $\frac{R}{n_1 n_2}$
  - still biased towards equal partition
  - no principled rationale behind this definition

denominator has its largest value when $n_1=n_2=n/2$

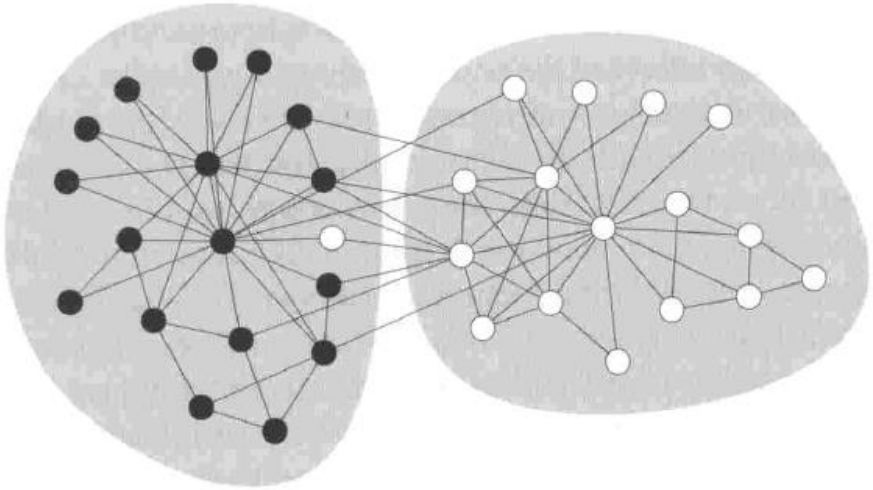
quality measure other than cut size

- a good division is one where there are fewer than expected such edges
- modularity (as the measure of assortative mixing)
- modularity maximization: look for the divisions that have the highest modularity scores
  - hard problem (takes exponential time)
  - heuristic algorithms are often used

simple modularity maximization

- analog of the Kernigham-Lin algorithm
  - starting from some initial (random) division
  - for each vertex check how much modularity would increase if it is moved to the other group
  - repeat the above process

  - identified communities are good
  - quite efficient : O(mn)



spectral modularity maximization (1)

- analogous to spectral graph partitioning

$$Q = \frac{1}{2m}\sum_{ij}(A_{ij} - \frac{k_i k_j}{2m})\delta(c_i, c_j) = \frac{1}{2m}\sum_{ij} B_{ij}\delta(c_i, c_j)$$

c_i: community to which vertex i belongs

δ(m,n): Kronecker delta

$$B_{ij} = A_{ij} - \frac{k_i k_j}{2m}$$

$$\sum_j B_{ij} = \sum_j A_{ij} - \frac{k_i}{2m}\sum_j k_j = k_i - \frac{k_i}{2m} 2m = 0$$

- division of a network into two parts

$$s_i = \begin{cases} +1 & \text{if vertex i belongs to group 1} \\ -1 & \text{if vertex i belongs to group 2} \end{cases}$$

$$\delta(c_i, c_j) = \frac{1}{2}(s_i s_j + 1)$$

$$Q = \frac{1}{4m}\sum_{ij} B_{ij}(s_i s_j + 1) = \frac{1}{4m}\sum_{ij} B_{ij} s_i s_j \qquad Q = \frac{1}{4m}\mathbf{s}^T \mathbf{B}\mathbf{s}$$

modularity matrix

spectral modularity maximization (2)

- find the value of **s** that maximize Q

$$Q = \frac{1}{4m}\mathbf{s}^T \mathbf{B}\mathbf{s}$$

elements of s are +1 or -1

- relax the constraints on **s**
  - elements of s can take arbitrary value
  - but its length has to be the same $\mathbf{s}^T\mathbf{s} = \sum_i s_i^2 = n$

$$\frac{\partial}{\partial s_i}\left[\sum_{jk} B_{jk}s_j s_k + \beta\left(n - \sum_j s_j^2\right)\right] = 0$$

$$\sum_j B_{ij}s_j = \beta s_i \qquad \mathbf{Bs} = \beta\mathbf{s}$$

s is one of the eigenvectors of the modularity matrix

spectral modularity maximization (3)

$$Q = \frac{1}{4m}\beta \mathbf{s}^T \mathbf{s} = \frac{n}{4m}\beta$$

- choose $\mathbf{s} = \mathbf{u}_1$ (eigenvector corresponding the largest eigenvalue of the modularity matrix)
- maximizing the product

$$\mathbf{s}^T \mathbf{u}_1 = \sum_i s_i [\mathbf{u}_1]_i \qquad s_i = \begin{cases} +1 & \text{if } [\mathbf{u}_1]_i > 0 \\ -1 & \text{if } [\mathbf{u}_1]_i < 0 \end{cases}$$

- <algorithm>
  1. calculate the eigenvector of the modularity matrix that corresponding to the largest eigenvalue
  2. assign vertices to communities according to the signs of the vector elements (positive/negative)

spectral modularity maximization (4)

- unlike Laplacian, modularity matrix is not sparse
  - finding the leading eigenvector:O(mn)
    - $O(n^3)$ for dense matrix
    - $O(n^2)$ for sparse matrix
  - by exploiting special properties of the modularity matrix, it is still possible to find the eigenvector in time $O(n^2)$ on a sparse network

division into more than two groups

- repeating bipartitioning
  - even if each bipartition is optimal, repeated bipartitioning may not be optimal
- consider the change ΔQ in the modularity of the entire network
  - bisecting a community c of size n_c

$$\Delta Q = \frac{1}{2m}\left[\frac{1}{2}\sum_{i,j\in c} B_{ij}(s_i s_j + 1) - \sum_{i,j\in c} B_{ij}\right]$$

$$= \frac{1}{4m}\left[\sum_{i,j\in c} B_{ij} s_i s_j - \sum_{i,j\in c} B_{ij}\right] = \frac{1}{4m}\sum_{i,j\in c}\left[B_{ij} - \delta_{ij}\sum_{k\in c} B_{ik}\right] s_i s_j$$

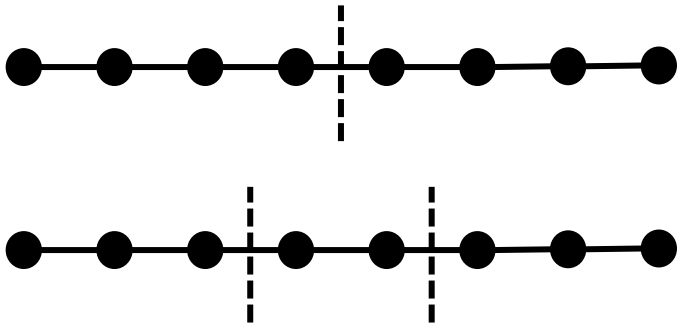
$$= \frac{1}{4m}\mathbf{s}^T \mathbf{B}^{(c)} \mathbf{s}$$

$$B_{ij}^{(c)} = B_{ij} - \delta_{ij}\sum_{k\in c} B_{ik}$$

$n_c \times n_c$ matrix

weakness of repeated bipartitioning

- repeated optimal bipartitioning may not be able to find optimal division



other modularity maximization methods

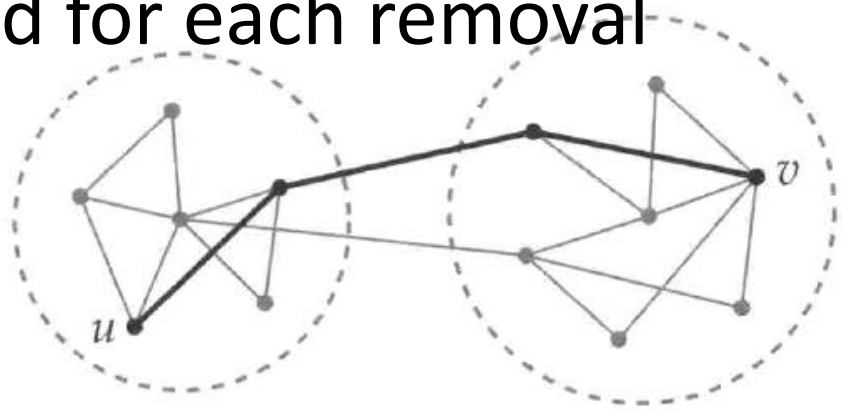
- simulated annealing
  - analogy with the physics of slow cooling of solids
  - ground state: the state of the lowest energy
  - pros: high quality cons: slow
- genetic algorithm
  - analogy with biological evolution
  - assign “fitness” to each of the population of different divisions
  - pros: high quality cons: slow
- greedy algorithm
  - bottom up : choose the merge of the biggest increase of modularity
  - pros: fast cons: moderate quality

other algorithms for community detection

- there is no universally agreed definition of communities
  - the previous discussion focus on modularity maximization, but there are other definitions
- some algorithms are introduced in the following slides
  - betweenness-based method
  - hierarchical clustering
- if you want to learn more, read the following article
  - “Community detection in graphs” by Santo Fortunato
  - Physics Reports, Vol. 486, Issues 3-5, pp.75-174 February 2010
  - http://www.sciencedirect.com/science/article/pii/S0370157309002841

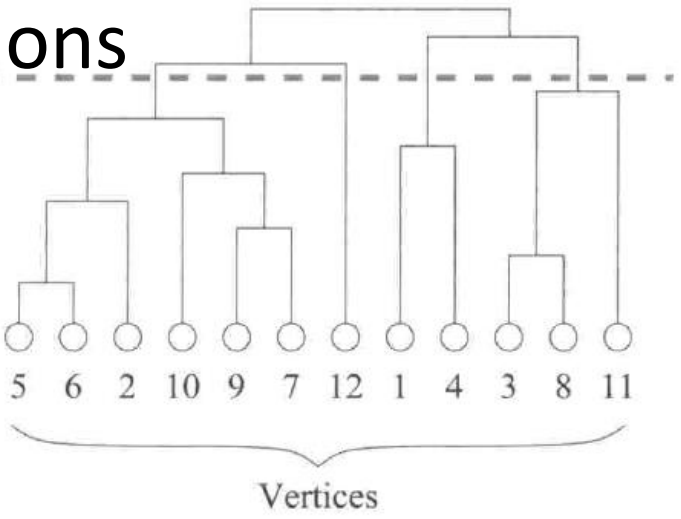
betweenness-based methods

- look for the edges that lie between communities, and remove them
- edge betweenness: the number of geodesic (shortest) paths that run along the edge
  - takes time of order O(n(m+n))
- recalculation is required for each removal



dendrogram

- root(top): all vertices in one group
- leaves(bottom): each vertex in a one-vertex group
- the algorithm generates the dendrogram from top to bottom
- selection from different divisions
  - coarse division (top)
  - fine division (bottom)

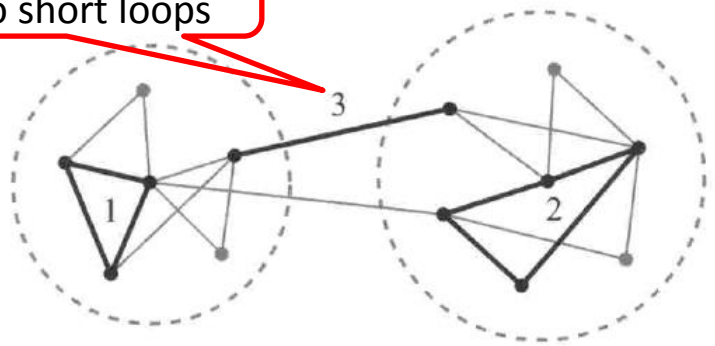


comments on betweenness-based method

- pros
  - hierarchical decomposition : dendrogram
- cons
  - slow: entire algorithm takes time O(mn(m+n))
- Radicchi's method
  - search for the edges that belong to short loops
  - faster : $O(n^2)$

$O(n^3)$ for sparse networks

"bridge" edges do not belong to short loops

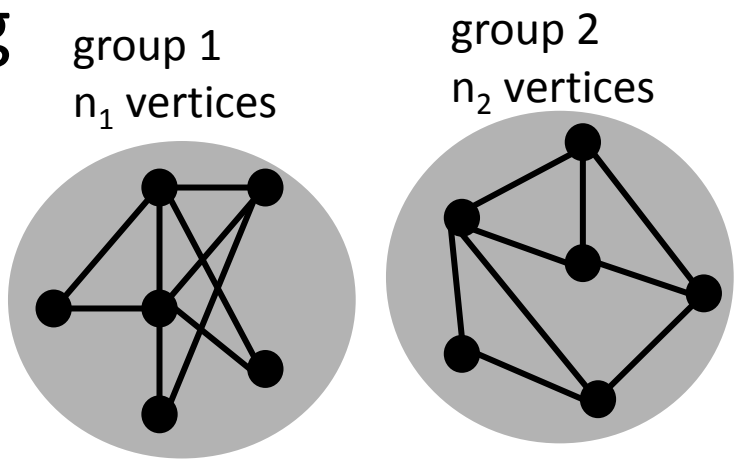


hierarchical clustering

- agglomerative (<-> divisive)
- define a measure of similarity, and join the most similar vertices to form groups
  - cosine similarity
  - correlation coefficient
  - Euclidean distance

similarity of vertices → similarity of groups

- there are $n_1 n_2$ pairs of vertices
- single-linkage clustering
  - similarity of the most similar pair
- complete-linkage clustering
  - similarity of the least similar pair
- average-linkage clustering
  - mean similarity of all pairs

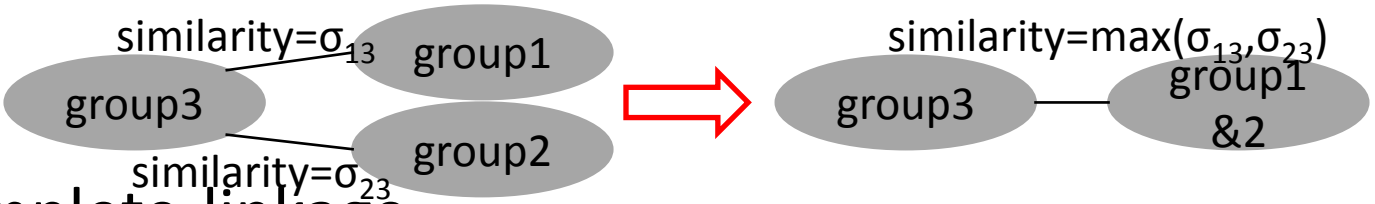


procedure of hierarchical clustering

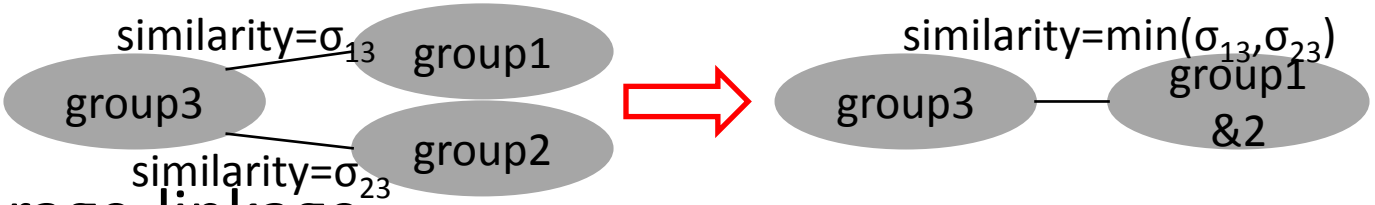
1. Choose a similarity measure and evaluate it for all vertex pairs.
2. Assign each vertex to a group of its own, consisting of just that one vertex. The initial similarities of the groups are simply the similarities of the vertices.
3. Find the pair of groups with the highest similarity and join them together into a single group.
4. Calculate the similarity between the new composite group and all others using one of the three methods (single-, complete-, or average-linkage clustering).
5. Repeat from step 3 until all vertices have been joined into a single group.

similarity of two groups after join

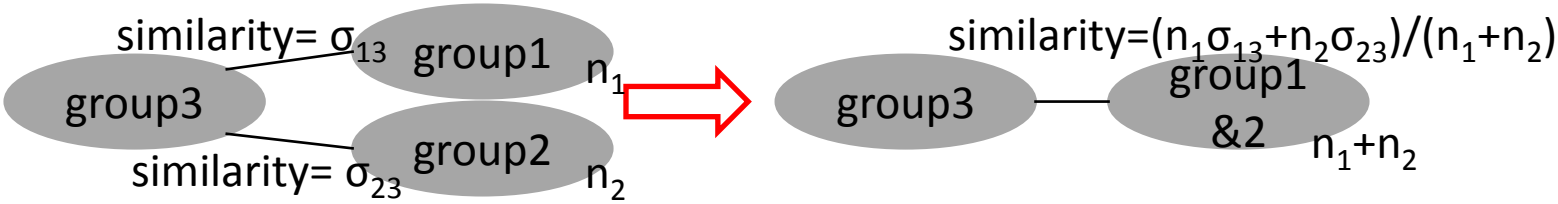
- single-linkage



- complete-linkage



- average-linkage



-

computational complexity of hierarchical clustering

- O(n) for recalculation of similarities
  - $O(n^2)$ for naive approach (recalculation + search for the biggest one)
  - O(nlogn) for using a heap (recalculation + storing in a binary heap)
- joining groups has to be repeated n-1 times
- total
  - $O(n^3)$ for naive implementation
  - $O(n^2 \log n)$ for using a heap

comments on hierarchical clustering

- results depend on which similarity measure one chooses and which linkage method
- good at picking out the cores of groups, but less good at assigning peripheral vertices to appropriate groups