School of Information Sciences
University of Pittsburgh

TELCOM2125: Network Science and Analysis

Konstantinos Pelechrinis

Spring 2015

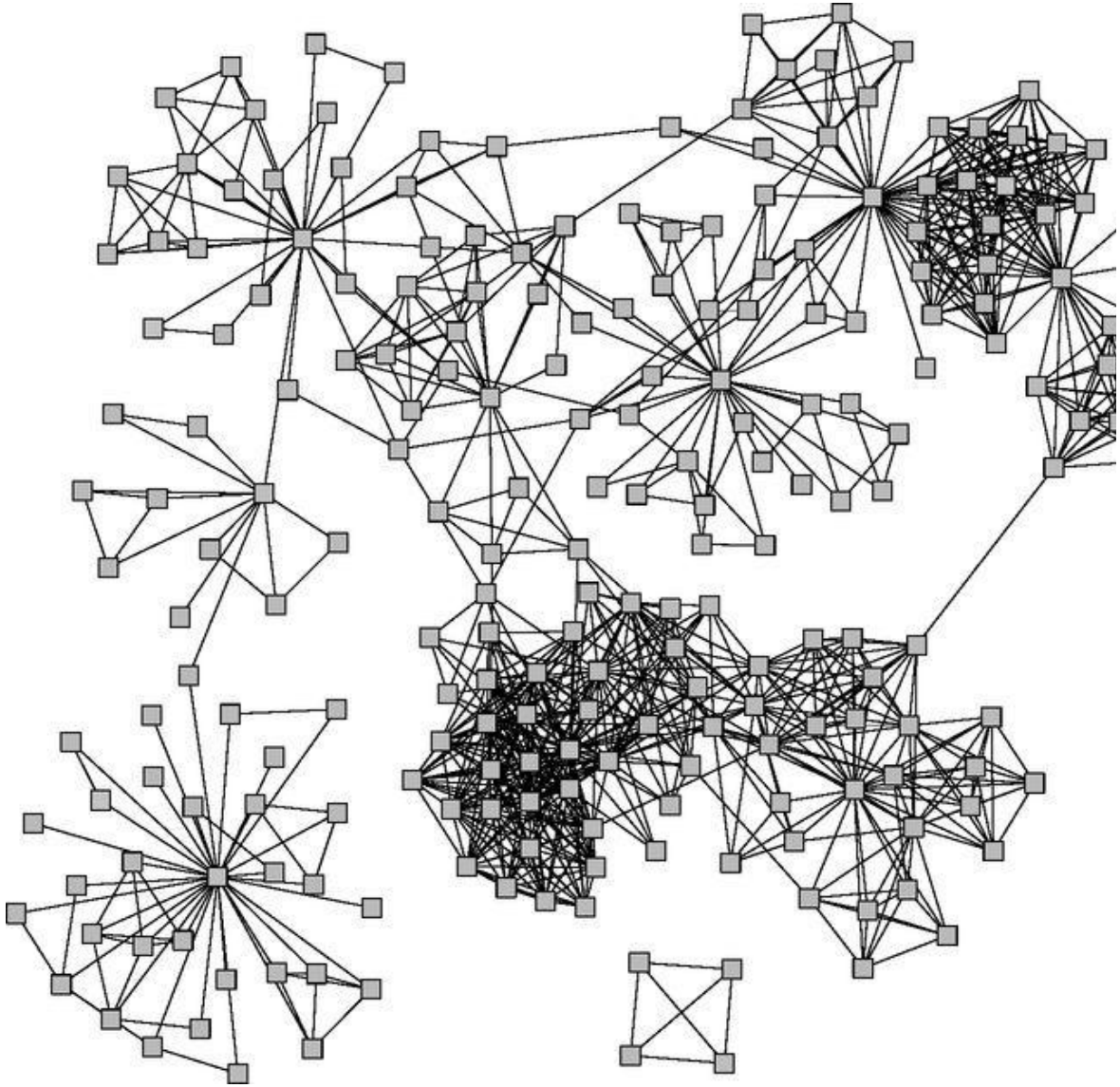
Part 4: Dividing Networks into Clusters

The problem

- **Graph partitioning and community detection refer to the division of the vertices in a graph based on the connection patterns of the edges**
  - The most common approach is to divide vertices in groups, such that the majority of edges connect members from the same group

- **Discovering communities in a network can be useful for revealing structure and organization in a network beyond the scale of a single vertex**

Visually



Number of groups

- **Depending on whether we have specified the number of groups we want to divide our network in we have two different problems:**
  - Graph partitioning
  - Community detection

- **Even though the general problem is the same, there are different algorithms that deal with each one of the problems**

Graph partitioning

- **In graph partitioning the number of groups k is pre-defined**

- **In particular, we want to divide the network vertices into k non-overlapping groups of given sizes such that the number of edges across groups are minimized**
  - Sizes can also be provided in approximation only
    - ✓ E.g., within specific range
  - E.g., divide the network nodes into two groups of equal size, such that the number of edges between them is minimized

- **Graph partitioning is useful in parallel processing of numerical solutions of network processes**

Community detection

- **In community detection neither the number of groups nor their sizes are pre-defined**

- **The goal is to find the *natural fault lines along which the network separates***
  - Few edges between groups and many edges within groups

- **Community detection is not as well-posed problem as graph partitioning**
  - What do we mean “many” and “few”?
  - Many different objectives → different algorithms

Graph partitioning vs Community detection

- **Graph partitioning is typically performed to divide a large network into smaller parts for faster/more manageable processing**

- **Community detection aims into understanding the structure of the network in a large scale**
  - Identifying connection patterns

- **Hence, graph partitioning always needs to output a partition (even if it is not good), while community detection is not required to provide an answer if no good division exists**

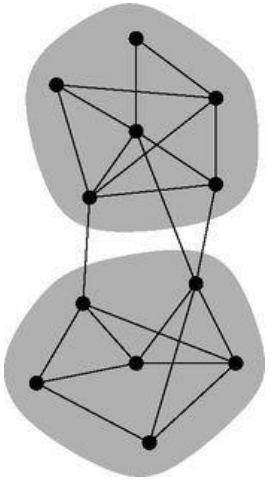
Why partitioning is hard ?

- **Graph partitioning is an easy problem to state and understand but it is not easy to solve**

- **Let's start with the simplest problem of dividing a network into two equally sized parts**
  - Graph bisection
  - Partitioning in an arbitrarily number of parts can be realized by repeated graph bisection
  - The number of edges between the two groups is called

  cut size

  ✓ Can be thought as a generalized min cut problem



Why partitioning is hard ?

- **An optimal solution would require to examine all the possible partitions of the network in two groups of sizes n_1 and n_2 respectively. This is:**

$$\frac{n!}{n_1!n_2!}$$

- **Such an algorithm would require exponential time and hence, is not appropriate for even moderately small networks**
  - We have to rely on approximation/heuristic algorithms
    - ✓ They require less time but the solution is not optimal (but acceptable)

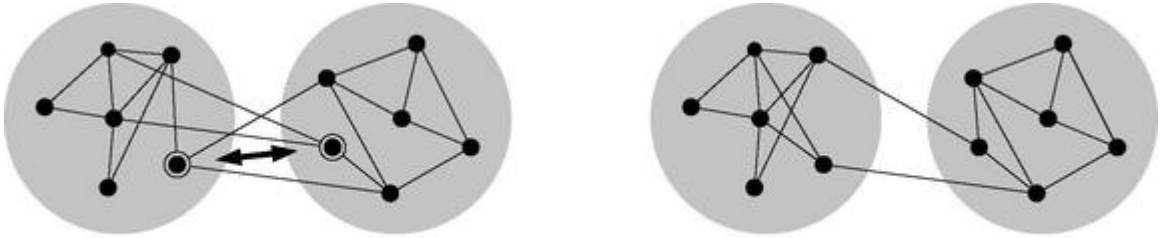
Kernighan-Lin algorithm

- **The Kernighan-Lin algorithm is a *greedy* heuristic for graph partitioning**
- **We begin with an initial (possibly random) partition of the vertices**
  - At each step we examine every pair of nodes (i,j), where i and j belong to different groups
    - ✓ If swapping i and j has the *smallest effect on the cut size* – i.e., maximum decrease or least increase – we swap i and j
  - We repeat the above process by considering all pairs across groups but not ones including vertices that have already been swapped
  - Once all swaps have been completed, we go through all the different states the network passed through and choose the state in which the cut size takes the smallest value

Kernighan-Lin algorithm

- **We can repeat this algorithm for many rounds**
  - At each round we start with the best division found in the previous round
  - If no improvement of the cut size occurs during the new round the algorithm terminates

- **Different initial assignments can give different final partitioning,**
  - We can run the algorithm several times and pick the best solution



Kernighan-Lin algorithm

- **Despite being straightforward Kernighan-Lin algorithm is slow**

- **At every round the number of swaps is equal to the size of the smaller of the groups → O(n)**

- **At every step of every round we have to check all the possible pairs of nodes that belong to different groups and we have not yet swapped → $O(n^2)$**
  - For each of these pairs we need to determine what is the change in the cut size if we swap them

Kernighan-Lin algorithm

- **If k_i^{same} are the number of edges that node i has with vertices belonging to the same group and k_i^{other} are the edges i has across groups, then the total change in the cut size if we swap i and j is:**

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

- **Evaluation of the above expression depends on the way we store the network**
  - Adjacent list: O(m/n)
  - With some tricks and with adjacent matrix: O(1)

- **Nevertheless, the overall running time is $O(n^3)$**

Spectral partitioning

- **Let us consider that the vertices are partitioned in groups 1 and 2**
  - The total number of edges between vertices of these groups is:

$$R = \frac{1}{2}\sum_{i,j} A_{ij}, \quad \text{where i and j belong to different groups}$$

- **Let's define:**

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

  - Then:

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

Spectral partitioning

- **Hence, we can now write R as:**

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

  - The summation now takes place over all i and j

- **We also have:** $\sum_{ij} A_{ij} = \sum_{i} k_i = \sum_{i} k_i s_i^2 = \sum_{ij} k_i \delta_{ij} s_i s_j$

- **Hence, we can write:**

$$R = \frac{1}{4}\sum_{ij}(k_i\delta_{ij} - A_{ij})s_i s_j = \frac{1}{4}\sum_{ij} L_{ij} s_i s_j \Rightarrow R = \frac{1}{4}\vec{s}^T L\vec{s}$$

  - L is the Laplacian matrix

Spectral partitioning

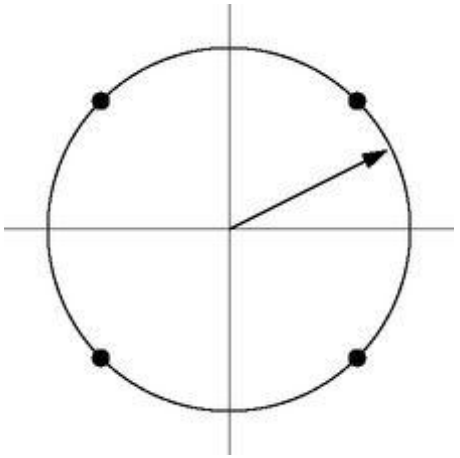
- The above optimization problem is the formal definition of the graph partitioning (bisection)
- The reason why it is hard to solve is the fact that the elements s_i cannot take any real value but only specific integer values (+/- 1)
- Integer optimization problems can be solved approximately using the *relaxation method*

Spectral partitioning

- **In the original integer problem the vector s would point to one of the 2^n corners of an n-dimensional hypercube centered at the origins**
  - The length of this vector is $\sqrt{n}$

- **We relax the above constraint by allowing the vector s to point to any direction as long as its length is still $\sqrt{n}$**

$$\sum_i s_i^2 = n$$



Spectral partitioning

- **A second constraint in the integer optimization problem is the fact that the number of elements in vector s that equal +1 or -1 needs to be equal to the desired sizes of the two groups:**

$$\sum_i s_i = n_1 - n_2 \Rightarrow \vec{1}^T \vec{s} = n_1 - n_2$$

  - We keep this constraint unchanged

- **So we have a minimization problem subject to two equality constraints**
  - Lagrange multipliers

Spectral partitioning

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

$$\Downarrow$$

$$L\vec{s} = \lambda\vec{s} + \mu\vec{1}$$

- The vector $(1,1,\ldots,1)^T$ is an eigenvector of L with eigenvalue 0. Hence, if we multiply the above equation on the left with (1,1,…,1) we have:

$$\mu = -\frac{n_1 - n_2}{n}\lambda$$

- If we define: $\vec{x} = \vec{s} + \frac{\mu}{\lambda}\vec{1} = \vec{s} - \frac{n_1 - n_2}{n}\vec{1}$ we finally get:

$$L\vec{x} = L\left(\vec{s} + \frac{\mu}{\lambda}\vec{1}\right) = L\vec{s} = \lambda\vec{s} + \mu\vec{1} = \lambda\vec{x}$$

Spectral partitioning

- **The last equations tells us that vector x is an eigenvector of the Laplacian, with eigenvalue λ**

- **Given that any eigenvector will satisfy this equation we will pick the one that minimizes R**
  - However, it cannot be the vector (1,1,…,1) since:

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

    - ✓ Vector x is orthogonal to vector (1,1,…,1)

- **So which eigenvector ?**

Spectral partitioning

- **We rewrite R as:**

$$R = \frac{1}{4}\vec{s}^T L\vec{s} = \frac{1}{4}\vec{x}^T L\vec{x} = \frac{1}{4}\lambda\vec{x}^T\vec{x} = \frac{1}{4}\lambda\left(\vec{s}^T\vec{s} + \frac{\mu}{\lambda}(\vec{s}^T\vec{1} + \vec{1}^T\vec{s}) + \frac{\mu^2}{\lambda^2}\vec{1}^T\vec{1}\right) =$$

$$= \frac{1}{4}\lambda\left(n - 2\frac{n_1 - n_2}{n}(n_1 - n_2) + \frac{(n_1 - n_2)^2}{n}n\right) = \frac{n_1 n_2}{n}\lambda$$

- **Hence, the cut size is proportional to the eigenvalue λ**
  - We pick the second lowest eigenvalue λ_2 (since λ=0 cannot be a solution as we saw) and then vector x is proportional to the eigenvector v_2
  - Finally vector s is given by:

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

This vector is ***relaxed***

Spectral partitioning

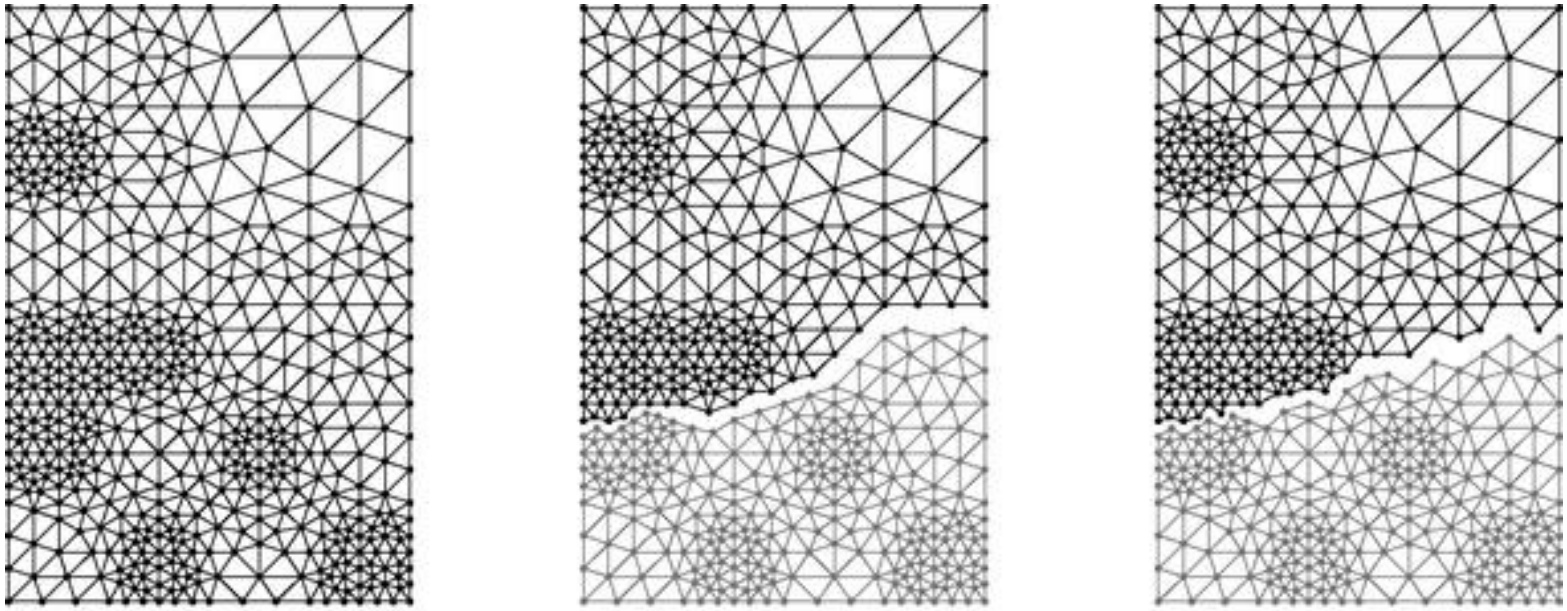
- **How do we transform the obtained vector s to one with values only +/- 1 ?**
  - We simply set the n_1 most positive values of the relaxed vector s equal to 1, and the rest equal to -1
  - Since vector x, and hence vector s, are proportional to the eigenvector v_2 of the Laplacian we finally have a very simple result
    - ✓ Calculate the eigenvector v_2 of the Laplacian and place the n_1 vertices with the most positive elements in v_2 to group 1 and the rest to group 2
    - ✓ When n_1 and n_2 are not equal, there are two ways of making the split. Which do we pick?

Spectral partitioning

- **The final steps are as follows:**
  - Calculate the eigenvector v_2 that corresponds to the second smallest eigenvalue λ_2 of the graph Laplacian
  - Sort the elements of the eigenvector in order from the largest to smallest
  - Put the vertices corresponding to the n_1 largest elements in group 1, the rest in group 2 and calculate the cut size
  - Put the vertices corresponding to the n_1 smallest elements in group 1, the rest in group 2 and calculate the cut size
  - Between these two divisions of the network, choose the one that gives the smaller cut size

Spectral partitioning

- **In general the spectral partitioning will find divisions of a network that have the right shape, but are not as good as the ones obtained by other algorithms**
- **However, the main advantage is the speed and hence its scalability**
  - O(mn) in sparse network



Algebraic connectivity

- **We have seen before that the second smallest eigenvalue of the graph Laplacian informs us whether the network is connected or not**

- **Now when the network is connected its value tells us how easy is to divide the network**
  - We saw that the cut size is proportional to λ_2
  - Small λ_2 means that the network has good cuts
    - ✓ Can be disconnected by removing a few edges

Community detection

- **Community detection is primarily used as a tool for understanding the large-scale structure of networks**

- **The number and size of groups are not specified**
  - Community detections aims into finding the *natural* division of the network into groups
    - ✓ The fault line along which the network is divided into groups is not well defined and so is our problem

- **Let's start with the simplest community detection problem instance**
  - We want to divide the network again into two groups but now we do not know the size of these groups

Community detection

- **One solution would be to perform graph bisection with one of the algorithms presented before for all combinations of n_1 and n_2 and pick the one with the smallest cut size**
  - This will not work (why?)

- **Another approach is to define the ratio cut partitioning:** $\frac{R}{n_1 n_2}$
  - Now instead of minimizing R, we want to minimize the above ratio
  - The denominator takes its maximum value when $n_1=n_2=0.5n$, while it diverges if one of the quantities is zero
  - Hence, the solution where n_1 or n_2 equals to zero is eliminated, but still the solution is biased (now towards equally sized groups)

Community detection

- **It has been argued that cut size is not a good metric for evaluating community detection algorithms**

- **A good division is one where edges across groups are less than the expected ones if edges were places in random**
  - It is not the total cut size that matters but how it compares with what we expect to see

- **Conventionally in the development of the community detection algorithms one considers the edges within groups**

Community detection

- **This approach is similar to the thinking developed for the assortativity coefficient and modularity**
  - Modularity takes high values when connections between vertices of the same type are higher than the number of connections expected at random
- **In the context of community detection, the vertices of the two groups can be considered to be of the same type**
  - Good divisions are the ones that have high values of the corresponding modularity
- **Community detection is also an NP problem**

Simple modularity maximization

- **One straightforward approach is an algorithm analogous to the Kernighan-Lin graph bisection**

- **We start with a random division of the network into equally sized groups**
  - At each step go through all *vertices* and compute the change in the modularity if this vertex was placed to the other group
  - Among all vertices examined swap groups to the one that has the *best* effect on modularity – i.e., largest increase or smallest decrease
  - We repeat the above process by considering only the vertices that we have not already swaped

Simple modularity maximization

- **As with the Kernighan-Lin algorithm, after we go through all vertices we go through all the states of the network and keep the one with the higher modularity**
  - We repeat the whole process starting with the best network that we identified in the previous round, until the modularity no longer improves

- **The time complexity of this algorithm is significantly less than the one of Kernighan-Lin algorithm**
  - In this case we go through single vertices and not all possible pairs
    - ✓ Complexity O(mn)

Spectral modularity maximization

- **The modularity matrix B is defined as:**

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

- **The modularity matrix has the following property:**

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

- **Let's consider the simplest case again, that is dividing the vertices into two groups**
  - We define again s_i similar to the case of spectral partitioning. Hence,

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

$$s_i = \begin{cases} +1, & i \quad in \quad group \quad 1 \\ -1, & i \quad in \quad group \quad 2 \end{cases}$$

Spectral modularity maximization

- **Using the above we can write the modularity of the network as:**

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

- **The above objective function is of the same form as the one for the spectral bisection**
  - We still have the constraint $\sum_i s_i^2 = n$
  - The only difference is that we do not have the constraint for the sizes in the groups
  - As before we solve the problem using the relaxation method
    - ✓ One Lagrange multiplier now

Spectral modularity maximization

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

$$\Downarrow$$

$$B\vec{s} = \beta\vec{s}$$

- **In other words vector s in this problem is an eigenvector of the modularity matrix**
  - Hence, $Q = \frac{1}{4m}\beta\vec{s}^T\vec{s} = \frac{n}{4m}\beta$
- **For maximum modularity, we pick vector s to be the eigenvector that corresponds to the maximum eigenvalue of the modularity matrix**

Spectral modularity maximization

- **For removing the relaxation we want to maximize: $s^T u_1$ (why?)**

- **We approximate vector s as following:**

$$s_i = \begin{cases} +1, & if\ [u_1]_i > 0 \\ -1, & if\ [u_1]_i < 0 \end{cases}$$

- **Modularity matrix is not sparse hence the running time can be high**
  - By exploiting properties of the modularity matrix we can still compute the eigenvector in $O(n^2)$ for a sparse network

Division into more than two groups

- **In the case of community detection we cannot simply continue dividing the output pairs of the community detection algorithms for two communities**
  - The modularity of the complete network does not break up into independent contributions from the separate communities

- **We must explicitly consider the change ΔQ in the modularity of the entire network upon further bisecting the community c of size n_c**

$$\begin{aligned}\Delta Q &= \frac{1}{2m}\left[\frac{1}{2}\sum_{i,j\in c} B_{ij}(s_i s_j + 1) - \sum_{i,j\in c} B_{ij}\right] \\ &= \frac{1}{4m}\left[\sum_{i,j\in c} B_{ij} s_i s_j - \sum_{i,j\in c} B_{ij}\right] = \frac{1}{4m}\sum_{i,j\in c}\left[B_{ij} - \delta_{ij}\sum_{k\in c} B_{ik}\right] s_i s_j \\ &= \frac{1}{4m}\mathbf{s}^T \mathbf{B}^{(c)} \mathbf{s},\end{aligned}$$

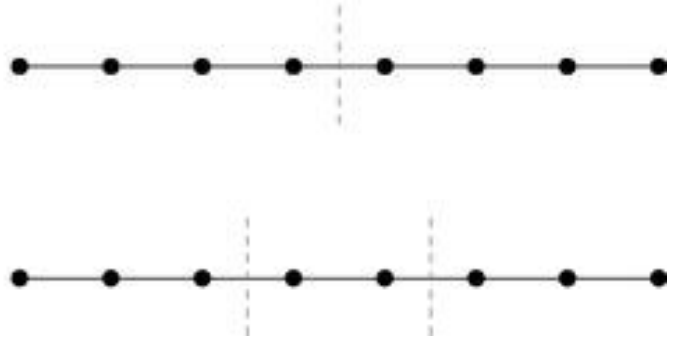
Division into more than two groups

- **The above problem has the same format as the spectral modularity maximization**
  - Hence, we find the leading eigenvector of $B^{(c)}$ and divide the network according to the signs of its elements

- **When do we stop this iterative process?**
  - If we are unable to find any division of a community that results in a positive change ΔQ in the modularity, we leave that community undivided
    - ✓ The leading eigenvector will have all elements with the same sign
  - When we have subdivided the network to the point where all communities are in this indivisible state, the algorithm terminates

Division into more than two groups

- **The repeated bisection is by no means perfect**



- **As we have mentioned we can even attempt to find directly the maximum modularity over divisions into any number of groups**
  - This can, in principle, find better divisions
    - ✓ More complicated and slower

Other modularity maximization methods

- **Simulated annealing**
- **Genetic algorithms**
- **Greedy algorithms**
  - We begin with every node belonging to a different community
  - At every step we combine the pair of groups whose amalgamation gives the biggest increase in modularity, or the smallest decrease if no choice gives an increase
  - Eventually all vertices are combined into a single group
  - We then trace back the states over which the network passed and we select the one with the highest value of modularity
  - Time complexity $O(n\log^2 n)$
    - ✓ The modularity achieved with greedy algorithms is slightly lower but it is much faster compared to the other two approaches

Betweenness-based method

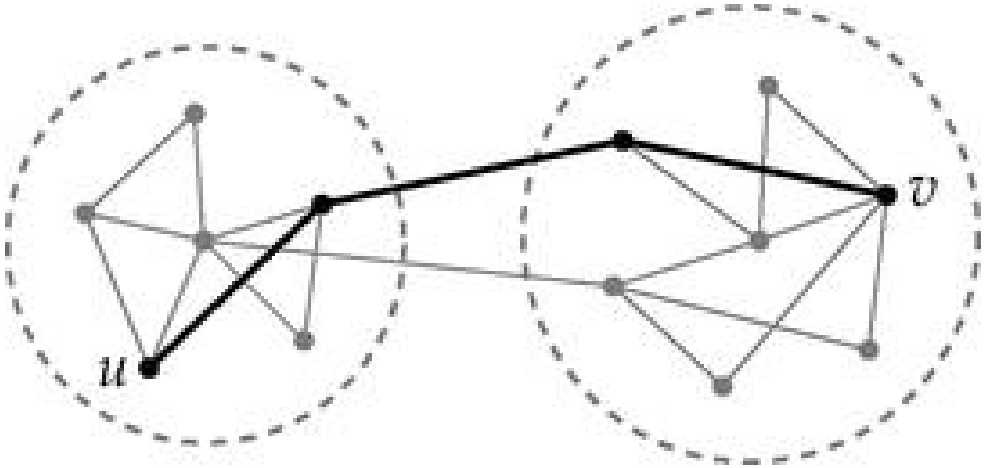
- **While graph partitioning is a problem well-defined objective function (i.e., minimize cut set) the same does not hold true for community detection**
  - Modularity maximization is only one of the possible ways to tackle the problem
  - Different objective functions can lead to different algorithms

- **One alternative to modularity maximization is to find edges that lie between communities**
  - Once we remove them we will be left with just the isolated communities

Betweenness-based method

- ***Edge betweenness*** **is the number of geodesic paths that go through an edge**

- **Edges that lie between communities are expected to have high betweenness because they will connect roughly all pairs of vertices between the communities**
  - Time to calculate edge betweenness is O(n(m+n))

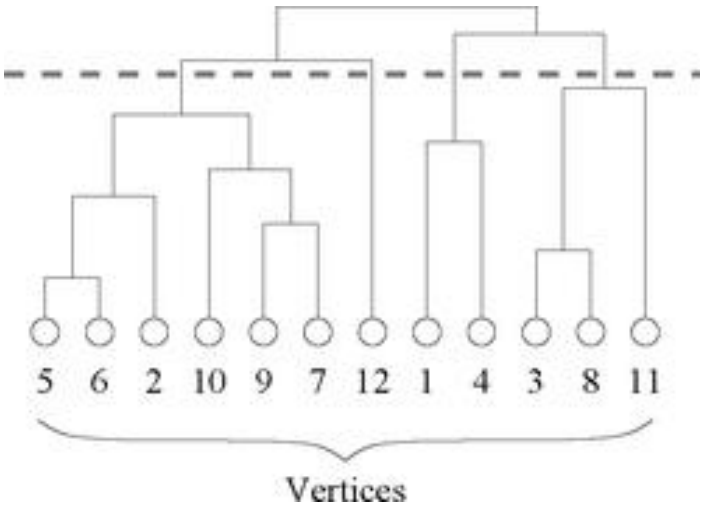


Betweenness-based method

- **We calculate the betweenness of each edge**
- **We find the edge with the highest score and remove it**
- **We recalculate the betweenness of the remaining edges, since geodesic paths will have changed**
- **Repeat the above process until we eventually have a completely disconnected network**
  - Singleton vertices

Betweenness-based method

- **The states of the network passes through the above procedure can be represented using a dendrogram**
- **Leaves represent the final state (i.e., singleton nodes) while the top of the tree represents the starting state (i.e., a single community)**
  - Horizontal cuts represent intermediate configurations
- **This algorithm gives a selection of different possible decompositions of the network**
  - Coarse (i.e., top of the tree) vs fine divisions (i.e., bottom of the tree)

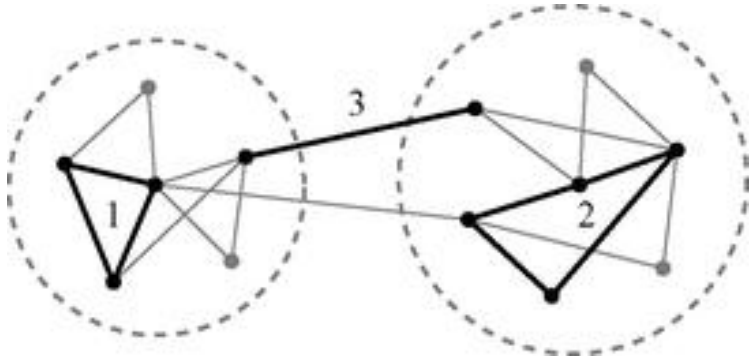


Betweenness-based method

- **The above algorithm gives very good results**
- **However it is one of the slowest algorithms for community detection**
  - $O(n^3)$ in a sparse network
- **The ability to get an entire dendrogram rather than a single decomposition can be useful in many cases**
  - Hierarchical decomposition

Betweenness-based method

- **A variation of the above algorithm is based on the observation that an edge that connects two components that are poorly connected with each other is unlikely to participate to many short loops**
  - Thus, one can identify the edges between communities by examining the number of short loops they participate in (rather than the betweenness)
  - Radicchi *et al.* found that examining for loop lengths 3 and 4 gave the best results



Betweenness-based method

- **This approach is an order of magnitude faster compared to the betweenness-based approach**
  - Computing short loops to which an edge belongs is a local operation

- **However, this approach works well when the network has a significant number of short loops to begin with**
  - Social networks do have many short loops, however other types of networks (e.g., technological or biological) might not have this property

Hierarchical clustering

- **The betweenness-based community detection provided as an output a dendrogram**
  - The dendrogram essentially decomposes the network into nested communities in a hierarchical fashion

- **There is an entire class of algorithms that are doing the same thing but operate using an agglomerative strategy rather than a divisive one**
  - Hierarchical clustering algorithms

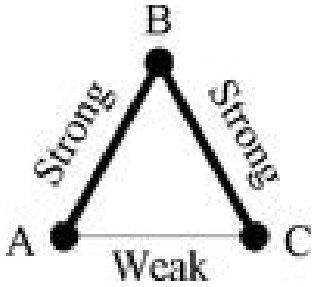
Hierarchical clustering

- **The basic idea behind hierarchical clustering is to initially consider every vertex as one community**
- **Then we define a metric of similarity between vertices, based on the network structure**
  - Then we join the most similar vertices to the same group
- **We can use any metric of structural equivalence as the similarity metric**
  - Cosine similarity, correlation coefficient between rows in the adjacent matrix, Euclidean distance between rows in the adjacent matrix etc.

Hierarchical clustering

- **Once we calculate the pairwise similarities, our goal is to group together the vertices that have the highest similarities**
  - This can lead to conflicts

- **The first step is to join the singleton pairs with the highest similarity to a group of size two**
  - Then we need to agglomerate groups with possibly more than one vertices
  - However, we have metrics for pairwise similarity not group similarity
  - How can we combine these vertex similarities to create similarities for the groups?



Hierarchical clustering

- **Consider two groups of n_1 and n_2 vertices respectively**
  - There are $n_1 n_2$ pairs of vertices among them (one node belongs to group 1 and the other to group 2)
- **Single-linkage clustering**
  - The similarity between the groups is defined as the maximum of the pairwise similarities
- **Complete-linkage clustering**
  - The similarity between the groups is defined as the smallest of the pairwise similarities

Hierarchical clustering

- **Average-linkage clustering**
  - The similarity between the groups is defined as the mean of the pairwise similarities

- **Single-linkage clustering is very lenient, while complete-linkage clustering is more stringent**
  - Average-linkage clustering is a more satisfactory definition and in between the two extremes, but it is rarely used in practice

Hierarchical clustering

- **The full hierarchical clustering method is:**
  - Choose a similarity metric and evaluate it for all vertex pairs
  - Assign each vertex to a group of its own, consisting of just that one vertex
    - ✓ The initial similarities of the groups are just the similarities of the vertices
  - Find the pair of groups with the highest similarity and join them together into a single group
  - Calculate the similarity between the new composite group and all others using one of the three methods above (single-, complete- or average-linkage clustering)
  - Repeat from step 3 until all vertices have been joined to a single group

Hierarchical clustering

- **Time complexity $O(n^2 \log n)$ using a heap**
  - With union/find and single-linkage clustering we can speed up to $O(n^2)$

- **Hierarchical clustering does not always work well**
  - It is able to find the cores of the groups, but tends to be less good at assigning peripheral vertices to appropriate groups
  - Peripheral nodes are left out of the agglomerative clustering until the very end
    - ✓ Hence, we have a set of tightly knit cores surrounded by a loose collection of single vertices or smaller groups
      - ○ However, even this might be valuable information for the network structure