Network Science (VU) (706.703)

Graph Partitioning and Community Detection

Denis Helic

ISDS, TU Graz

December 5, 2018

Outline

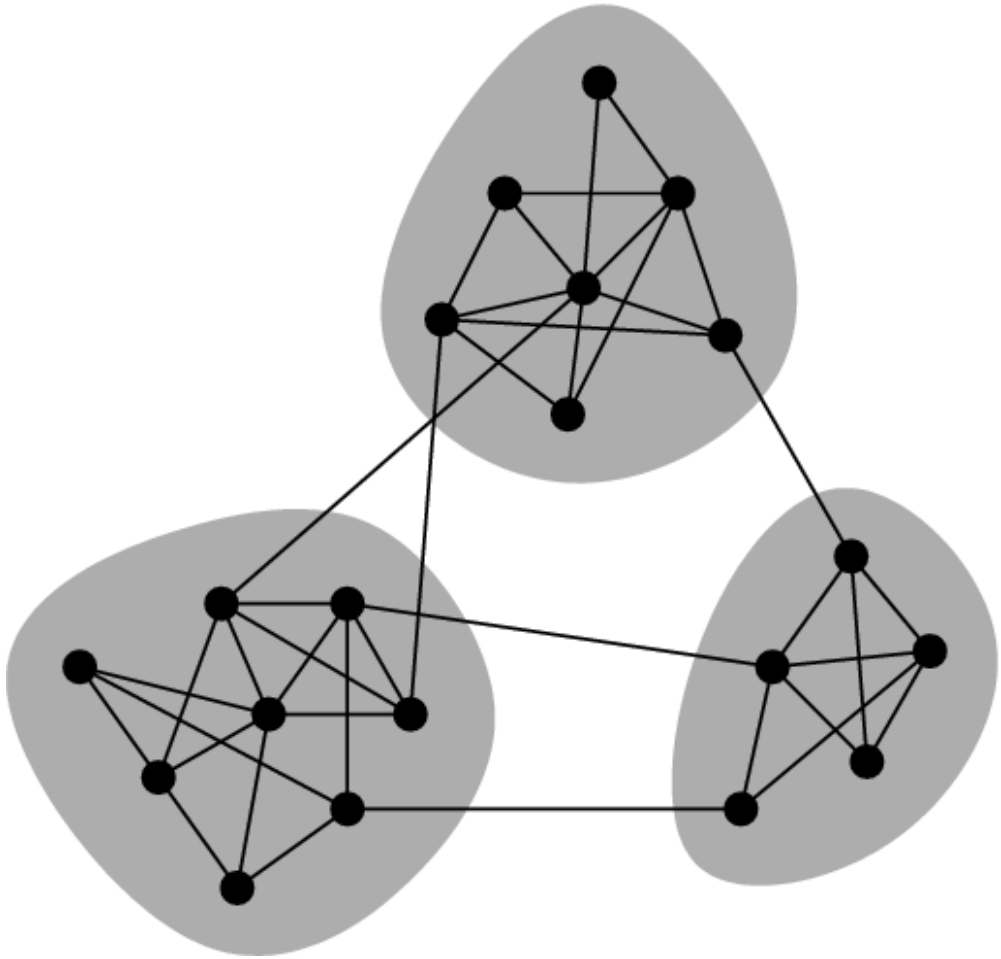
Slides

Slides on stochastic block models are partially based on slides by Prof. Aaron Clauset from University of Colorado at Boulder.

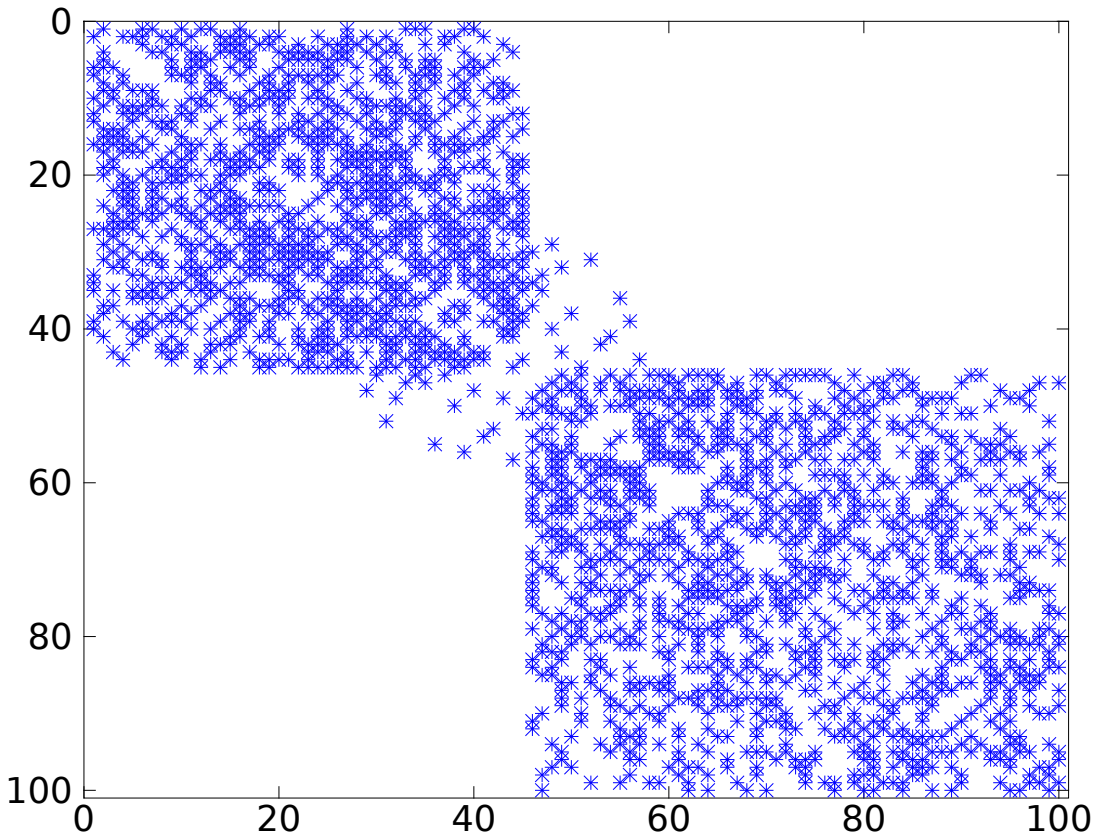
Dividing networks

- Graph partitioning and community detection refer to the division of nodes into groups, clusters, or communities
- Network internal criteria
- According to the pattern of links in the network
- External criteria are also possible, but we do not discuss them here
- Most commonly, the goal is to divide nodes so that the groups have many links inside groups and only a few between groups

Dividing Networks



Dividing Networks



Dividing networks

- Groups of nodes are of interest in many different areas
  - World Wide Web
  - Citation networks
  - Social networks
  - Biological, metabolic networks, etc.

Groups

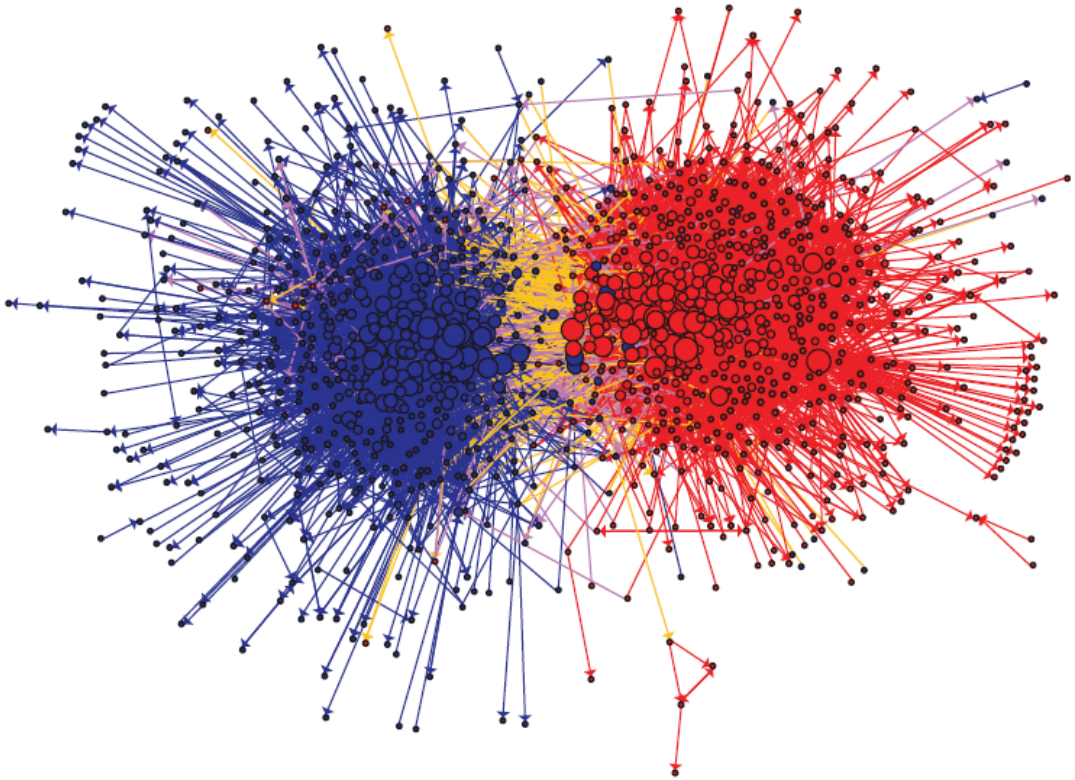
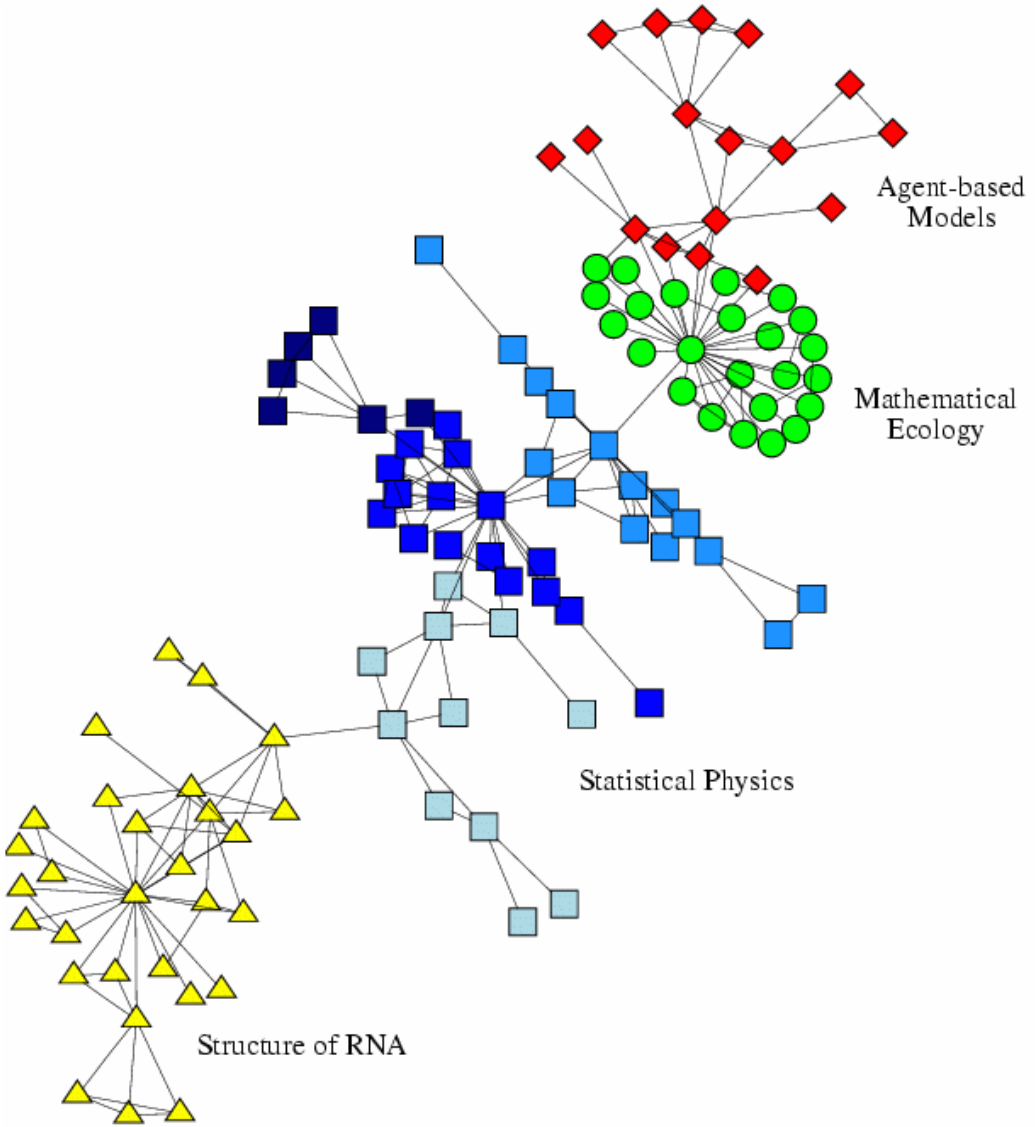


Figure: Political Blogosphere: US 2004 Elections [Adamic and Glance]

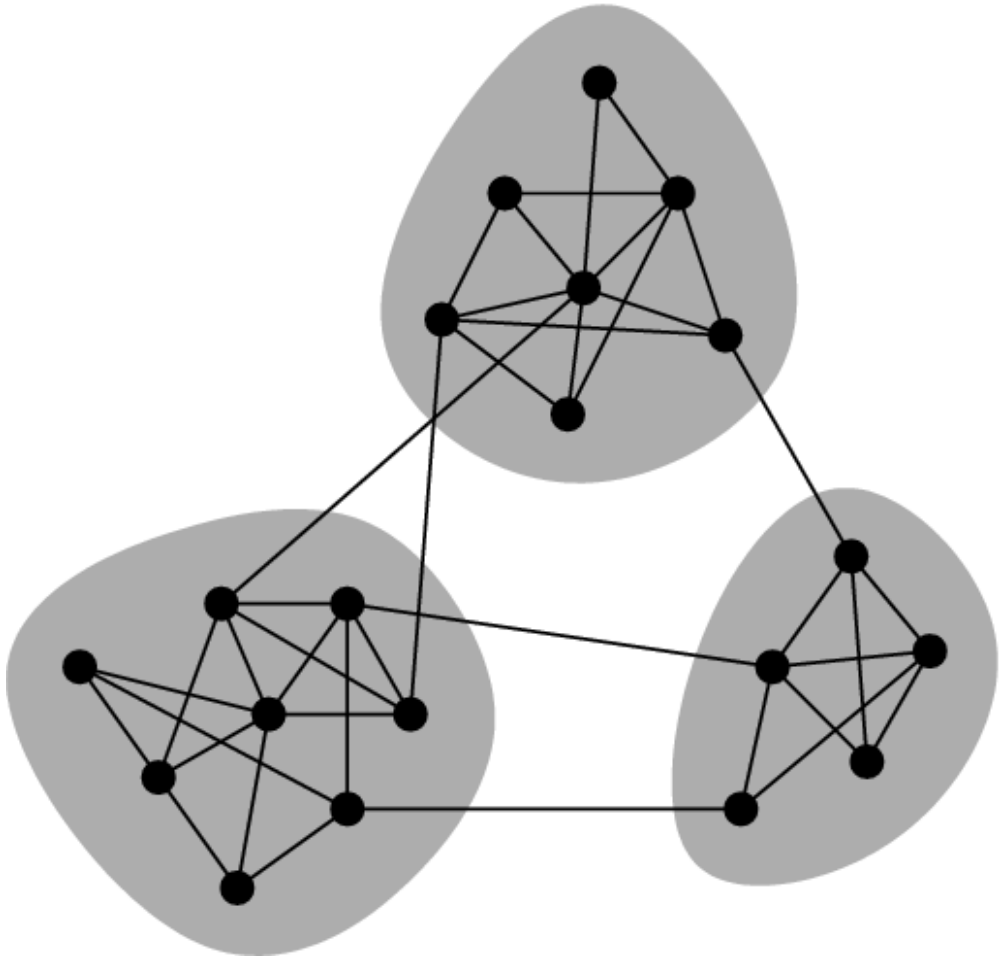
Groups


Agent-based
Models
Mathematical
Ecology
Statistical Physics
Structure of RNA

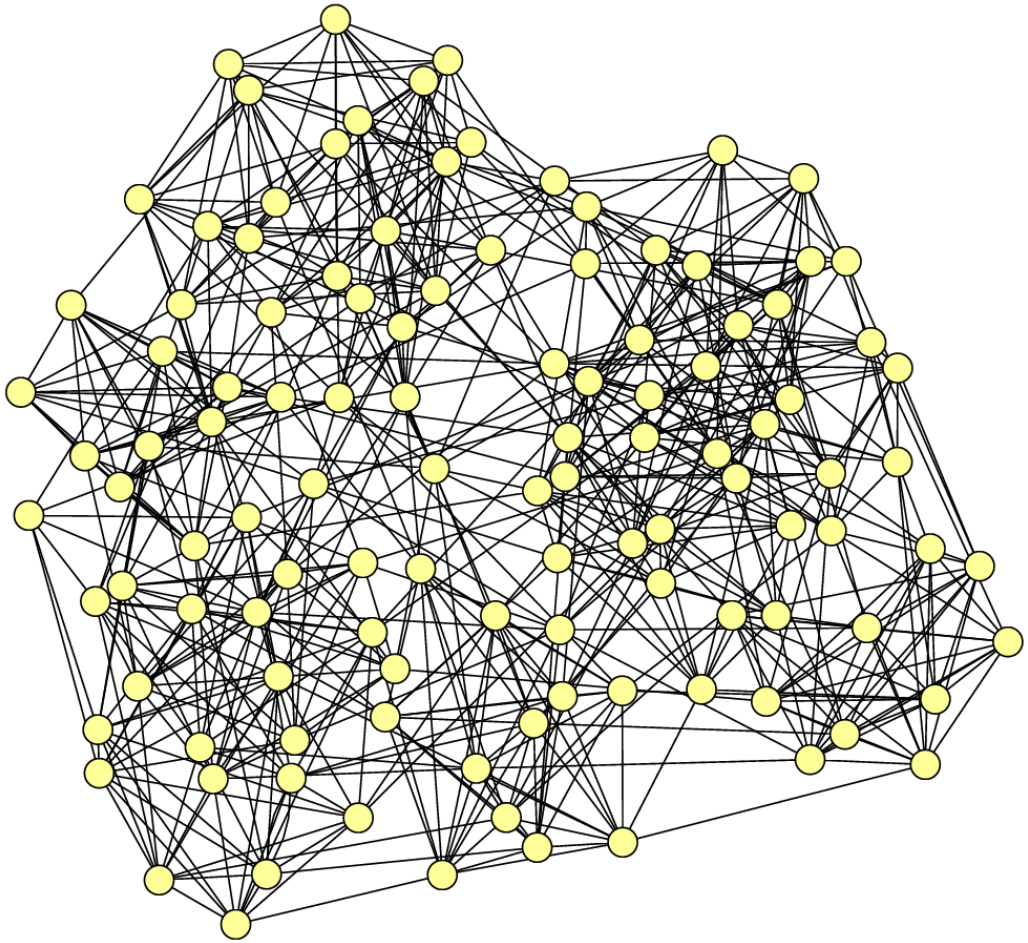
Dividing networks

- How to approach network division?
- If we have some external information (e.g. political party) the division is easy
- If networks are small visual identification is sometimes possible

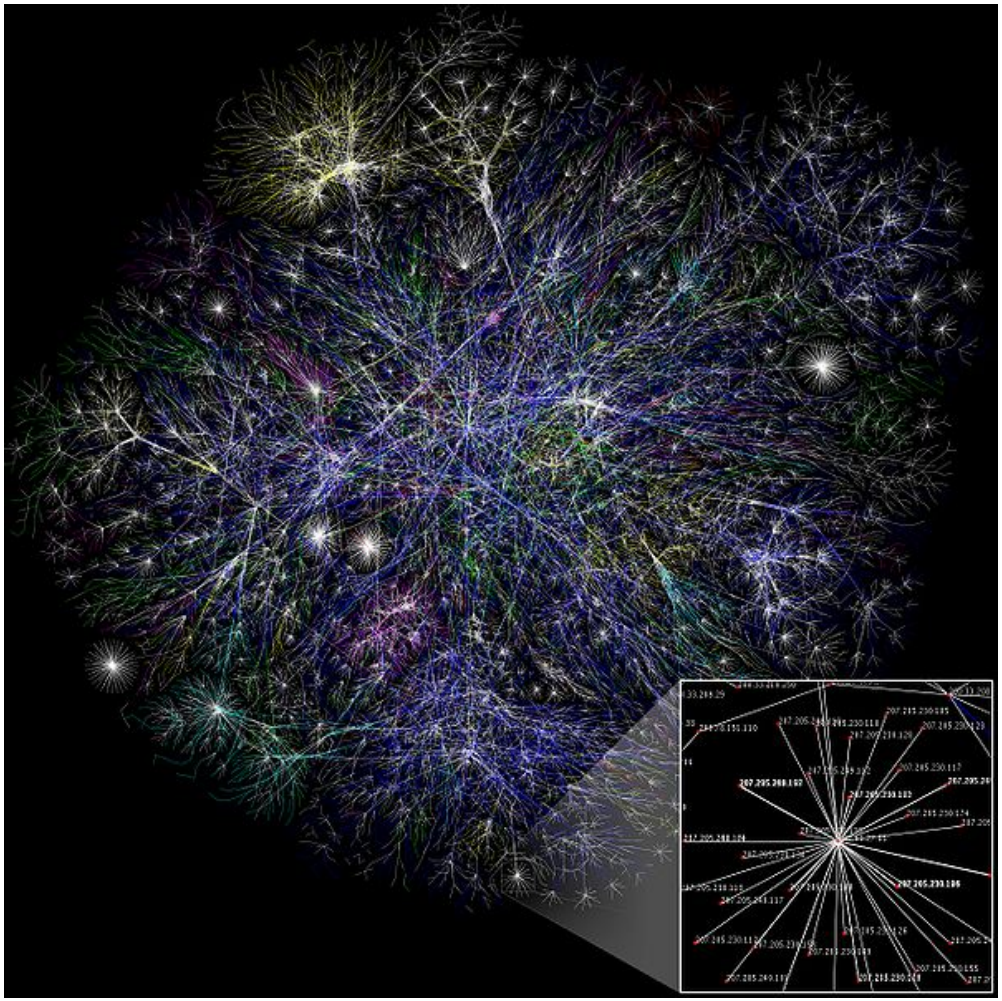
Dividing networks



Dividing networks



Dividing networks



Dividing networks

- However, we are interested in large networks
- Apply **algorithms**
- There are several reasons why we want to divide a network into groups or clusters
- However, there are two general reasons and that leads to two corresponding types of algorithms
- Two types are graph partitioning and community detection

Graph partitioning

- Graph partitioning is a classical problem in computer science
- It is the problem of dividing nodes into a given number of non-overlapping groups of given sizes such that the number of links between groups is minimized
- The important point: the number and the sizes of groups are fixed
- Sometimes the sizes are fixed within a range but they are still fixed
- A prototypical example is dividing a network into two groups of equal size such that the number of links between them is minimized

Graph partitioning

- A typical example from computer science is parallel computation
- Nodes are computation tasks and links represent data exchange between tasks
- There is a fixed number of CPUs
- We want to balance the workload between CPUs → equal group sizes
- We want to minimize the transfer of data between CPUs → minimize the number of links between groups

Community detection

- The other type of network division is the community detection
- It differs from graph partitioning in that the number and size of the groups is not specified
- Instead they are determined by the network itself
- The goal of community detection is to find the natural lines along which a network separates
- The sizes of the groups may vary widely from one group to another

Community detection

- The most common use of community detection is **understanding** of the network data
- E.g in social networks social communities
- E.g. in the Web clusters of related Web pages
- In metabolic networks functional units within the network
- Community detection is a less well-posed problem than graph partitioning

Community detection

- Community detection aims to find the natural divisions of a network into groups
- There are “many” links within groups and “few” links between groups
- How much is “many” or “few”?
- In summary, the main difference between graph partitioning and community detection
- The number and size of the groups is specified in graph partitioning and unspecified in community detection

Graph bisection

- The simplest graph partitioning problem is the division of a network into two parts
- Graph bisection
- We minimize the number of links (cut size) between two parts
- We can partition the graph into more than two parts by successive bisection of one or both parts of the first bisection
- Naive approach for bisection: look through all (exhaustive search) possible divisions and choose the one with the smallest cut size

How difficult is graph partitioning?

- How many ways to divide a network with n nodes into two groups of n_1 and n_2 nodes
- How many ways to choose n_1 (or n_2) nodes from n nodes
- The number of combinations without repeating and without replacement and without regard to their order is given by binomial coefficients

Binomial coefficients

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} \tag{1}$$

How difficult is graph partitioning?

- For graph bisection: $n_1 + n_2 = n$

$$\binom{n}{n_1} = \frac{n!}{n_1!(n-n_1)!} = \frac{n!}{n_1!n_2!} = \binom{n}{n_2} \tag{2}$$

How difficult is graph partitioning?

- We can approximate the factorials using Stirling's formula

$$n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n \tag{3}$$

$$\begin{aligned}\frac{n!}{n_1!n_2!} &= \frac{\sqrt{2\pi n}\left(\frac{n}{e}\right)^n}{\sqrt{2\pi n_1}\left(\frac{n_1}{e}\right)^{n_1}\sqrt{2\pi n_2}\left(\frac{n_2}{e}\right)^{n_2}} \\ &= \frac{1}{\sqrt{2\pi}}\frac{n^{(n+\frac{1}{2})}}{n_1^{(n_1+\frac{1}{2})}n_2^{(n_2+\frac{1}{2})}}\end{aligned} \tag{4}$$

How difficult is graph partitioning?

- E.g. if we want to bisect the graph into two parts of equal size $n_1 = n_2 = \frac{n}{2}$ the number of different ways is approximately:

$$\frac{1}{\sqrt{2\pi}} \frac{n^{(n+\frac{1}{2})}}{\left(\frac{n}{2}\right)^{\frac{n+1}{2}} \left(\frac{n}{2}\right)^{\frac{n+1}{2}}} = \frac{2^{(n+\frac{1}{2})}}{\sqrt{\pi n}} \tag{5}$$

- Thus, the amount of time to look through all divisions go up exponentially with the size of the network

Heuristics for graph partitioning

- Therefore we fall back to using heuristics that find approximate but acceptable results
- There are three approaches to such heuristics
  1. Greedy algorithms such as Kernighan-Lin algorithm or hierarchical clustering
  2. Optimization of some "reasonable" global criteria such as cut size, spectral clustering, or modularity
  3. Model-based methods that fit a probabilistic model for a network with communities

Kernighan-Lin algorithm

- One of the simplest and best known heuristic algorithms for the graph bisection problem
- Brian Kernighan: http://en.wikipedia.org/wiki/Brian_Kernighan
- We start by dividing nodes into two groups of the required sizes in any way we like
- For instance, we can divide nodes randomly
- Then for each pair of nodes (i, j) that lie in different groups we calculate how much the cut size would change if i and j change groups

Kernighan-Lin algorithm

- Among all pairs we find the pair that reduces the cut size by the largest amount
- If no pair reduces the cut size than we pick a pair which increases the cut size by the smallest amount
- Then we swap those nodes
- The process is repeated with the restriction that each node can be moved only once
- Thus, on the second step we consider all nodes and their pairs except the two nodes that have been swapped on the first step

Kernighan-Lin algorithm

- The algorithm proceeds until eventually there are no pairs left to be swapped
- When all swaps have been completed we go back through every state and pick the one in which the cut size takes the smallest value
- Then the entire process is repeated starting each time with the best division from the last round
- We continue until no improvement in the cut size occurs
- The best division from the last round is the result of the algorithm

Kernighan-Lin algorithm

- Note that if the initial assignment is random the algorithm may not give the same answer if it is run twice on the same network
- Probabilistic algorithm if the initial assignment is random
- For this reason you might want to run the algorithm a couple of times and compare the results
- The heuristic works very well for practical purposes

Complexity of Kernighan-Lin algorithm

- The primary disadvantage of Kernighan-Lin algorithm is that is slow
- The size of the groups lie between $\frac{n}{2}$ and n
- In the worst case there is $O(n)$ swaps in a single round
- For each swap we have to examine all pairs of nodes from different groups
- In the worst case: $\frac{n}{2} \times \frac{n}{2} = \frac{n^2}{4} = O(n^2)$
- For each of these we need to determine the change in the cut size

Complexity of Kernighan-Lin algorithm

- When node i moves from one group to another any links connecting to nodes in the current group become links between the groups after the swap
- Similarly, any links that i has to nodes in the other group become within-group links except for the link to node j, which is swapped with i
- To evaluate this in the adjacency list format we need to go through lists for node i and j
- On average they will be m/n long, and thus this step has $O(m/n)$
- The total time: $O(n \times n^2 \times m/n) = O(mn^2)$
- On a sparse network this is $O(n^3)$ and on a dense network $O(n^4)$

Spectral clustering: formalization

- We consider a network with n nodes and m links
- We divide network into two groups: g_1 and g_2
- The cut size of the division is given by:

$$R = \frac{1}{2} \sum_{i,j \text{ in different groups}} A_{ij} \tag{6}$$

Spectral clustering: formalization

- Let us define a vector $\mathbf{s}$ of quantities s_i, which represent the division of the network:

$$s_i = \begin{cases} 1 \text{ if node } i \text{ belongs to } g_1 \\ -1 \text{ if node } i \text{ belongs to } g_2 \end{cases} \tag{7}$$

How to calculate R?

- Lets calculate $s_i - s_j$
- First i and j may be from the same group: $s_i - s_j = 0$
- From different groups: $s_i - s_j = 2$ or $s_i - s_j = -2$, and $(s_i - s_j)^2 = 4$
- Laplacian quadratic form: $\mathbf{s}^T \mathbf{L}\mathbf{s} = \sum_{(i,j)\in E}(s_u - s_j)^2 = 4R$
- Thus, $R = \frac{1}{4}\mathbf{s}^T \mathbf{L}\mathbf{s}$

Spectral clustering as an optimization problem

- We want to minimize R
- Thus, it is now an optimization problem
- We want to find $\mathbf{s}$ that minimizes R, i.e. the cut size
- As seen previously this is a hard problem

Spectral clustering as an optimization problem

- What is hard in practice is that s_i can not take just any values
- They are restricted to special values ± 1
- If they were allowed to take any values then we could just differentiate to find the minimum
- This suggests a possible approximate approach

Spectral clustering as an optimization problem

- Suppose we let s_i to take any values but subject to certain constraints
- We find approximate values but they might be "good" enough
- This is a so-called *relaxation method* in mathematical optimization
- How should we define the constraints?

Relaxation

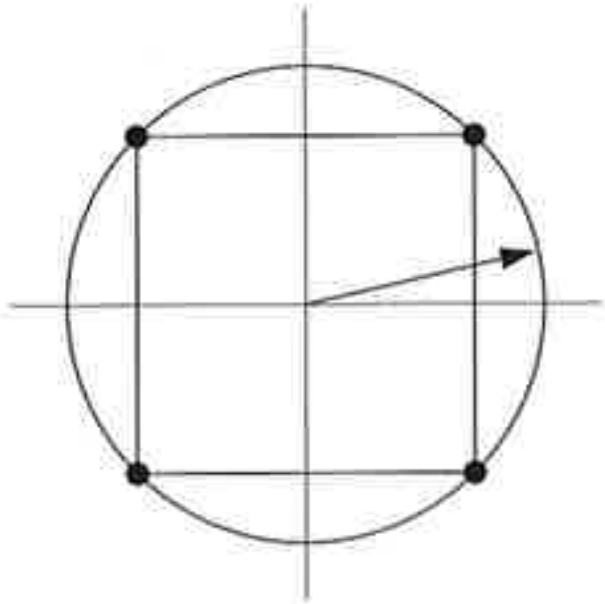
- Let us first take a look at the constraints if s_i can take only on values ± 1
- The first constraint determines the direction and the length of the vector $\mathbf{s}$
- $\mathbf{s}$ is a vector in an n dimensional Euclidean space
- $\mathbf{s}$ always points to one of the 2^n corners of a n-dimensional hypercube centered on the origin
- $\mathbf{s}$ always has the same length, which is $\sqrt{n}$

Relaxation

- Let us now relax the direction constraint and allow $\mathbf{s}$ to point in any direction
- We will keep the length constraint, i.e. the length of $\mathbf{s}$ remains $\sqrt{n}$
- It would not make much sense to allow the length to vary
- In that case, the minimization problem would have a trivial solution $\mathbf{s} = 0$
- Thus, we drop the direction constraint and keep the length constraint

$$\sum_i s_i^2 = n \tag{8}$$

Relaxation



Relaxation

- The second constraint on the vector $\mathbf{s}$ is on the number of $+1$ and -1
- These numbers must equal the sizes of the groups g_1 and g_2
- Let us denote these sizes with n_1 and n_2

$$\sum_i s_i = n_1 - n_2 \tag{9}$$

$$\mathbf{1}^T s = n_1 - n_2 \tag{10}$$

- We keep this second constraint unchanged

Constrained optimization: Lagrange multipliers

- Original objective function that we want to minimize: $\sum_{ij} L_{ij} s_i s_j$
- This function is subject to constraint: constrained optimization
- Typically solved by the method of Lagrange multipliers

$$\begin{aligned} \text{Objective function:} \quad f(\mathbf{s}) &= \sum_{ij} L_{ij} s_i s_j \\ \text{Subject to:} \quad \sum_i s_i^2 &= n \\ \text{Subject to:} \quad \sum_i s_i &= n_1 - n_2 \end{aligned}$$

Lagrange multipliers

- For each constraint we need one Lagrange multiplier, e.g. λ and 2μ
- Lagrange formulation of the optimization problem will be a new objective function that is a function of $\mathbf{s}$, λ and μ

$$L(\mathbf{s}, \lambda, \mu) = \sum_{ij} L_{ij} s_i s_j + \lambda(n - \sum_i s_i^2) + 2\mu((n_1 - n_2) - \sum_i s_i) \qquad (11)$$

Constrained optimization

- To minimize L we find s, λ and μ that make its gradient 0
- $\bigtriangledown L = 0$:

$$\begin{aligned} \frac{\partial L}{\partial s_i} &= 0, \forall i \\ \frac{\partial L}{\partial \lambda} &= 0 \\ \frac{\partial L}{\partial \mu} &= 0 \end{aligned}$$

Constrained optimization

- $\frac{\partial L}{\partial \lambda} = 0$ and $\frac{\partial L}{\partial \mu} = 0$ give back the constraints

$$\frac{\partial L}{\partial s_i} = 2\sum_j L_{ij} s_j - 2\lambda s_i - 2\mu = 0 \tag{12}$$

$$\sum_j L_{ij} s_j = \lambda s_i + \mu \tag{13}$$

$$\mathbf{Ls} = \lambda \mathbf{s} + \mu \mathbf{1} \tag{14}$$

Solution

- Recall that $\mathbf{1}$ is an eigenvector of $\mathbf{L}$ with the eigenvalue 0 so that $\mathbf{L} \cdot \mathbf{1} = 0$
- Let us multiply the last equation on the left with $\mathbf{1}^T$ and use $\mathbf{1}^T s = n_1 - n_2$:

$$\lambda(n_1 - n_2) + \mu n = 0 \tag{15}$$

$$\mu = -\frac{n_1 - n_2}{n}\lambda \tag{16}$$

Solution

- Let us define a new vector $\mathbf{x}$

$$\mathbf{x} = \mathbf{s} + \frac{\mu}{\lambda}\mathbf{1} = \mathbf{s} - \frac{n_1 - n_2}{n}\mathbf{1} \tag{17}$$

$$\mathbf{s} = \mathbf{x} + \frac{n_1 - n_2}{n}\mathbf{1} \tag{18}$$

Solution

- Let us calculate $\mathbf{Lx}$, using again $\mathbf{L} \cdot \mathbf{1} = 0$

$$\mathbf{Lx} = \mathbf{L}(\mathbf{s} + \frac{\mu}{\lambda}\mathbf{1}) = \mathbf{Ls} = \lambda\mathbf{s} + \mu\mathbf{1} = \lambda(\mathbf{s} + \frac{\mu}{\lambda}\mathbf{1}) = \lambda\mathbf{x} \tag{19}$$

- Thus, $\mathbf{x}$ is an eigenvector of $\mathbf{L}$

Solution

- Any eigenvector satisfies the last equation
- We should choose the eigenvector which minimizes R
- Can we take $\mathbf{1}$

$$\mathbf{1}^T\mathbf{x} = \mathbf{1}^T\mathbf{s} - \frac{n_1 - n_2}{n}\mathbf{1}^T\mathbf{1} = (n_1 - n_2) - \frac{n_1 - n_2}{n}n = 0 \qquad (20)$$

- $\mathbf{x}$ is orthogonal to $\mathbf{1}$ $\rightarrow$ we must pick another one

Solution

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

$$\begin{aligned}
\mathbf{s}^T\mathbf{L}\mathbf{s} &= (\mathbf{x} + \frac{n_1 - n_2}{n}\mathbf{1})^T\mathbf{L}\mathbf{s} = ((\mathbf{x}^T + \frac{n_1 - n_2}{n}\mathbf{1}^T)\mathbf{L})\mathbf{s} \\
&= (\mathbf{x}^T\mathbf{L} + \frac{n_1 - n_2}{n}\mathbf{1}^T\mathbf{L})\mathbf{s} = (\mathbf{x}^T\mathbf{L})\mathbf{s} = \mathbf{x}^T(\mathbf{L}\mathbf{s}) \\
&= \mathbf{x}^T\mathbf{L}\mathbf{x} = \lambda\mathbf{x}^T\mathbf{x} \qquad (21)
\end{aligned}$$

Solution

$$
\begin{aligned}
\mathbf{x}^T\mathbf{x} &= (\mathbf{s} - \frac{n_1 - n_2}{n}\mathbf{1})^T(\mathbf{s} - \frac{n_1 - n_2}{n}\mathbf{1}) \\
&= (\mathbf{s}^T - \frac{n_1 - n_2}{n}\mathbf{1}^T)(\mathbf{s} - \frac{n_1 - n_2}{n}\mathbf{1}) \\
&= \mathbf{s}^T\mathbf{s} - \frac{n_1 - n_2}{n}\mathbf{s}^T\mathbf{1} - \frac{n_1 - n_2}{n}\mathbf{1}^T\mathbf{s} + (\frac{n_1 - n_2}{n})^2\mathbf{1}^T\mathbf{1} \\
&= n - \frac{n_1 - n_2}{n}(n_1 - n_2) - \frac{n_1 - n_2}{n}(n_1 - n_2) + (\frac{n_1 - n_2}{n})^2 n \\
&= n - \frac{(n_1 - n_2)^2}{n} \\
&= \frac{4n_1n_2}{n}
\end{aligned}
\tag{22}
$$

Solution

$$\begin{aligned} R &= \frac{1}{4}\mathbf{s}^T\mathbf{L}\mathbf{s} \\ &= \frac{1}{4}\lambda\mathbf{x}^T\mathbf{x} \\ &= \frac{n_1 n_2}{n}\lambda \end{aligned} \tag{23}$$

- Thus, the cut size is proportional to the eigenvalue λ
- We want to minimize $R \rightarrow$ we should choose the smallest allowed eigenvalue

Eigenvalues of the graph Laplacian

- The vector $\mathbf{1}$ is always an eigenvector of $\mathbf{L}$ with eigenvalue 0
- There are no negative eigenvalues, thus this is the lowest eigenvalue
- Convention: $\lambda_1 \leq \lambda_2 \leq ... \leq \lambda_n$
- We always have $\lambda_1 = 0$

Eigenvalues of the graph Laplacian

- However, we can not take λ_1, and should choose λ_2
- The second eigenvalue of the Laplacian is called *algebraic connectivity*
- Thus, we take as $\mathbf{x}$ the $\mathbf{v}_2$ corresponding to λ_2
- Cut size is proportional to algebraic connectivity, which is a direct measure of how easy is to divide a network
- We might recover $\mathbf{s}$ by

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

Solution

- Vector $\mathbf{s}$ is however subject to further constraints, i.e. its elements take on values ± 1 and there are exactly n_1 elements with value 1 and exactly n_2 elements with value -1
- These constraints prevent $\mathbf{s}$ from taking the values from the last equation
- Let us do the best and choose $\mathbf{s}$ to be as close as possible to the ideal solution
- The ideal solution: $\mathbf{s}^T\mathbf{s} = n$

Solution

- Thus, we should try to maximize:

$$\mathbf{s}^T(\mathbf{x} + \frac{n_1 - n_2}{n}\mathbf{1}) = \sum_i s_i(x_i + \frac{n_1 - n_2}{n}) \tag{24}$$

- The maximum of this expression is achieved by assigning $s_i = +1$ for the nodes with most positive values of $(x_i + \frac{n_1 - n_2}{n})$ and $s_i = -1$ for the remainder
- The most positive values of $(x_i + \frac{n_1 - n_2}{n})$ are the most positive values of x_i, which are the most positive elements of $\mathbf{v}_2$

Solution

- Thus, we calculate the eigenvector $\mathbf{v}_2$ and place n_1 nodes with the most positive elements in group 1 and the rest in group 2
- It is arbitrary which group we call group 1 and which group 2
- Thus, if the sizes of two groups are different then we can make cut by taking n_1 most positive elements or n_2 most positive elements for group 1
- We can make both splits and calculate the cut size and then take the division which has a smaller cut size

Spectral clustering algorithm

- The final algorithm
  1. Calculate the eigenvector $\mathbf{v}_2$ corresponding to λ_2
  2. Sort the elements of the eigenvector in descending order
  3. Take n_1 nodes corresponding to the n_1 largest elements in group 1, the rest in group 2 and calculate R
  4. Take n_1 nodes corresponding to the n_1 smallest elements in group 1, the rest in group 2 and calculate R
  5. Between those two divisions choose one that gives the smaller cut size

Properties of spectral clustering

- In comparison to e.g. Kernighan-Lin spectral clustering produces typically similar division but the cut size is slightly worse
- This is typical of spectral method
- It tends to find divisions that have the right general shape but are perhaps not as good as those returned by the other methods
- However, the advantage of the spectral method is its speed

Complexity of spectral clustering

- The complexity is dominated by the calculation of the eigenvector $\mathbf{v}_2$
- This has complexity $O(mn)$
- On a sparse network this is $O(n^2)$
- On a dense network this is $O(n^3)$
- The majority of real-world networks are sparse
- Kernighan-Lin has the complexity of $O(n^3)$ on sparse networks and $O(n^4)$ on dense networks

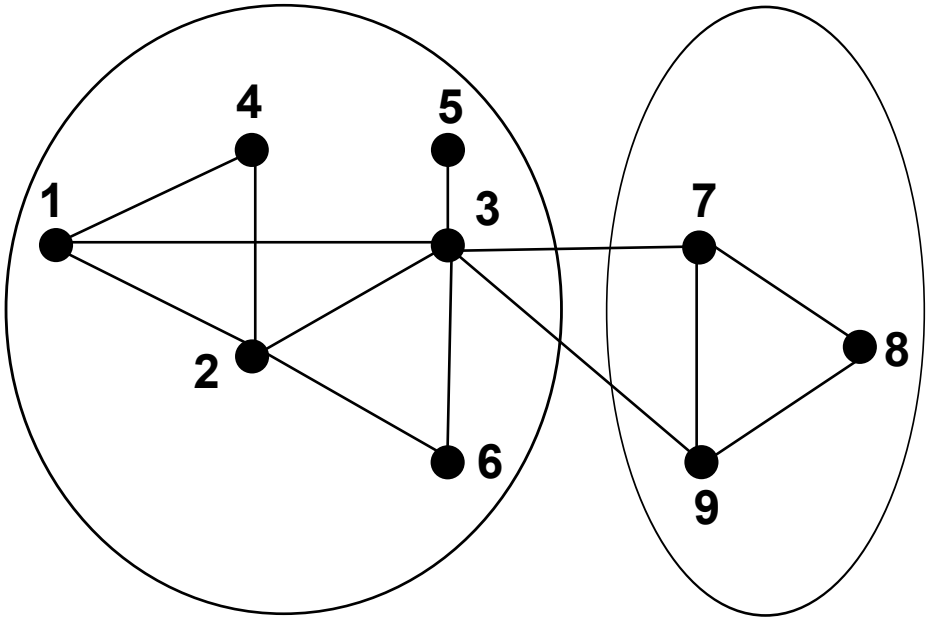
Further applications of spectral clustering

- We can apply the same approach to discover two communities
- Community detection algorithm
  1. Calculate the eigenvector $\mathbf{v}_2$ corresponding to λ_2
  2. Sort the elements of the eigenvector in descending order
  3. Take nodes corresponding to the positive elements in group 1, the rest in group 2

Further applications of spectral clustering

- We can apply the same approach to general community detection
- Community detection algorithm
  1. Calculate the eigenvector $\mathbf{v}_2$ corresponding to λ_2
  2. Sort the elements of the eigenvector in descending order
  3. Take nodes between large gaps in element values in corresponding groups

Example



Alternatives to cut size measure

- For community detection cut size minimization has some disadvantages
- We want to discover the communities such that the cut size is minimal
- But minimal cut size is 0
- The optimal division is then to not divide the network at all but put all the nodes into a single group
- An alternative strategy would be to focus on a different measure of the quality of division

Alternatives to cut size measure

- A good division of the network is not merely one in which there are few links between communities
- On the contrary, a good division is one where there are *fewer than expected* of such links
- If we find a division in which there are few links between communities, but nonetheless the number of these links is what we expect anyway than we did not find anything significant
- It is not the total cut size that matters but how that cut size compares to what we expect to see

Alternatives to cut size measure

- In fact, the conventional idea is that we consider the number of links within groups and not the number of links between groups
- The two approaches are equivalent since every link that lies within a group does not lie between groups
- Thus, we need a measure that quantifies how many links lie within groups relative to the number of such links expected on the basis of chance
- We already introduced the idea of assortative mixing and a measure that quantifies it
- **Modularity**
- Thus, we want to look for division that have the highest modularity

Simple example

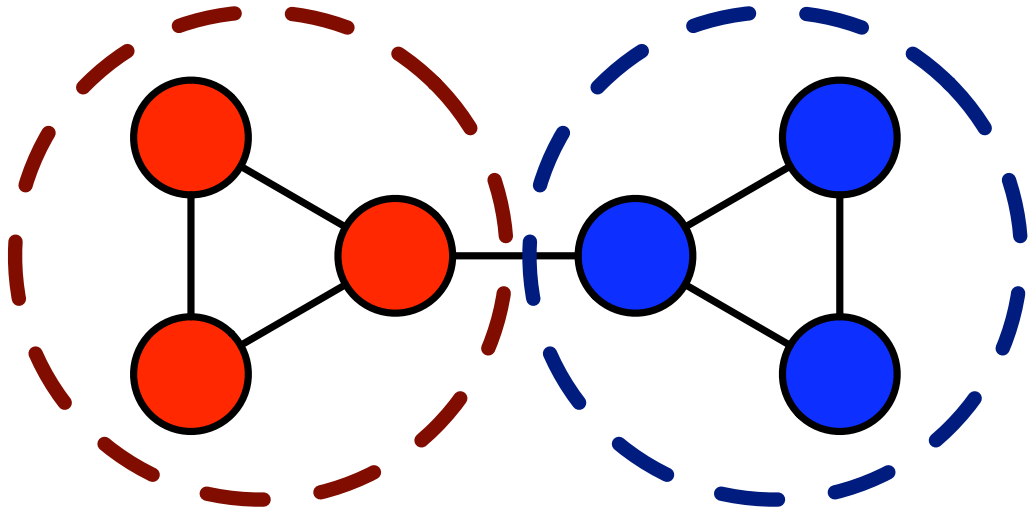


Figure: “Good” modularity

Simple example

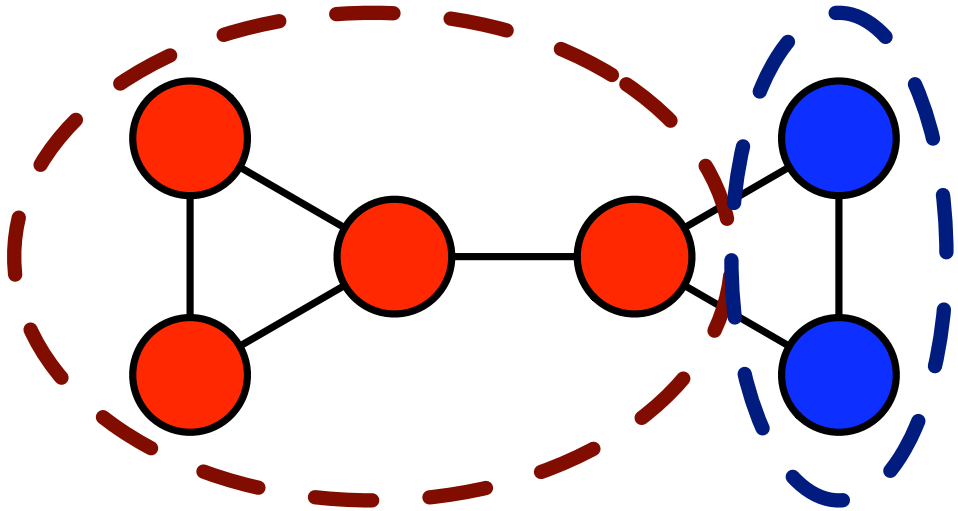


Figure: “Bad” modularity

Simple modularity maximization

- Analog to Kernighan-Lin
- We divide the network into two communities starting from some initial (random) division
- The algorithm considers each node in the network and calculates how much the modularity would change if that node is moved into the other group
- It then selects among all nodes the one node which the highest increase (or smallest decrease) in the modularity
- Then it repeats the process with the important constraint that a node once moved can not be again moved in the same round
- In this version we do not swap nodes, but just move a single node from one community to another (we do not need to keep the sizes of the communities constant)

Simple modularity maximization

- When all nodes have been moved exactly once we go back over all the states and select the state with the highest modularity
- That is the input for the next round
- We keep repeating the process until no further improvement in the modularity occurs
- In practice this gives very good results

Complexity of simple modularity maximization

- The node moving algorithm is quite efficient
- At each step of the algorithm we have to evaluate the change in the modularity due to movement of $O(n)$ nodes
- Each such evaluation requires again $O(m/n)$ in the adjacency list form
- This is repeated n times in one round of algorithm
- Total time: $O(n \times m/n \times n) = O(mn)$
- This is $O(n^2)$ on a sparse and $O(n^3)$ on a dense network

Spectral modularity maximization: formalization

- Is there an analog of the spectral clustering for modularity maximization?

$$Q = \frac{1}{2m} \sum_{ij} (A_{ij} - \frac{k_i k_j}{2m}) \delta(c_i, c_j) = \frac{1}{2m} \sum_{ij} B_{ij} \delta(c_i, c_j) \tag{25}$$

$$B_{ij} = A_{ij} - \frac{k_i k_j}{2m} \tag{26}$$

Spectral modularity maximization: formalization

- Note the following property of B_{ij}

$$\sum_j B_{ij} = \sum_j A_{ij} - \frac{k_i}{2m} \sum_j k_j = k_i - \frac{k_i}{2m} 2m = 0 \tag{27}$$

- Similarly $\sum_i B_{ij} = 0$

Spectral modularity maximization: formalization

- Let us first consider the division of a network in just two parts
- We introduce the division vector $\mathbf{s}$ as before

$$s_i = \begin{cases} 1 \text{ if node } i \text{ belongs to } g_1 \\ -1 \text{ if node } i \text{ belongs to } g_2 \end{cases}$$

Spectral modularity maximization: formalization

- Note that the quantity $\frac{1}{2}(s_i s_j + 1)$ is 1 if i and j are in the same group and 0 otherwise, so that:

$$\delta(c_i, c_j) = \frac{1}{2}(s_i s_j + 1) \tag{28}$$

$$Q = \frac{1}{4m}\sum_{ij} B_{ij}(s_i s_j + 1) = \frac{1}{4m}\sum_{ij} B_{ij} s_i s_j \tag{29}$$

$$Q = \frac{1}{4m}\mathbf{s}^T \mathbf{B}\mathbf{s} \tag{30}$$

Constrained optimization

- The form of equation is very similar to cut size for spectral clustering
- The constraint on $\mathbf{s}$ determines the direction and the length of the vector
- I.e. we have as before the following two constraints
  1. $\mathbf{s}$ always points to one of the 2^n corners of a n-dimensional hypercube centered on the origin
  2. $\mathbf{s}$ always has the same length, which is $\sqrt{n}$
- However, we do not have constraints on the size of the division

Relaxation method

- Again, let us relax the direction constraint and allow $\mathbf{s}$ to point in any direction
- Also, we will keep the length constraint, i.e. the length of $\mathbf{s}$ remains $\sqrt{n}$

$$\sum_i s_i^2 = n \tag{31}$$

Lagrange multipliers

- Original objective function that we want to maximize: $\sum_{ij} B_{ij} s_i s_j$
- This function is subject to constraint: we use again Lagrange multipliers

$$\begin{aligned} \text{Objective function:} \quad & f(\mathbf{s}) && = \sum_{ij} B_{ij} s_i s_j \\ \text{Subject to:} \quad & \textstyle\sum_i s_i^2 && = n \end{aligned} \tag{32}$$

Lagrange multipliers

- Now we need a single Lagrange multiplier, e.g. β
- Lagrange formulation of the optimization problem will be a new objective function that is a function of $\mathbf{s}$ and β

$$L(\mathbf{s}, \beta) = \sum_{ij} B_{ij} s_i s_j + \beta(n - \sum_i s_i^2) \tag{33}$$

Solution

- To maximize L we find $\mathbf{s}$ and β that make $\bigtriangledown L = 0$:

$$\begin{aligned} \frac{\partial L}{\partial s_i} &= 0, \forall i \\ \frac{\partial L}{\partial \beta} &= 0 \end{aligned} \tag{34}$$

Solution

- $\frac{\partial L}{\partial \beta} = 0$ gives back the constraints

$$\frac{\partial L}{\partial s_i} = 2\sum_j B_{ij}s_j - 2\beta s_i = 0 \tag{35}$$

$$\sum_j B_{ij}s_j = \beta s_i \tag{36}$$

$$\mathbf{Bs} = \beta \mathbf{s} \tag{37}$$

Solution

- In other words $\mathbf{s}$ is one of the eigenvectors of the modularity matrix

$$Q = \frac{1}{4m}\mathbf{s}^T\mathbf{B}\mathbf{s} = \frac{1}{4m}\beta\mathbf{s}^T\mathbf{s} = \frac{n}{4m}\beta \tag{38}$$

- For maximum modularity we should choose $\mathbf{s}$ to be the vector $\mathbf{u}_1$ that corresponds to the largest eigenvalue of the modularity matrix

Solution

- As before, vector $\mathbf{s}$ is however subject to further constraints, i.e. its elements take on values ± 1
- These constraints prevent $\mathbf{s}$ from taking the values from $\mathbf{s} = \mathbf{u}_1$
- Let us do the best and choose $\mathbf{s}$ to be as close as possible to the ideal solution
- Thus, we should maximize the product

$$\mathbf{s}^T \mathbf{u}_1 = \sum_i s_i [\mathbf{u}_1]_i \tag{39}$$

Solution

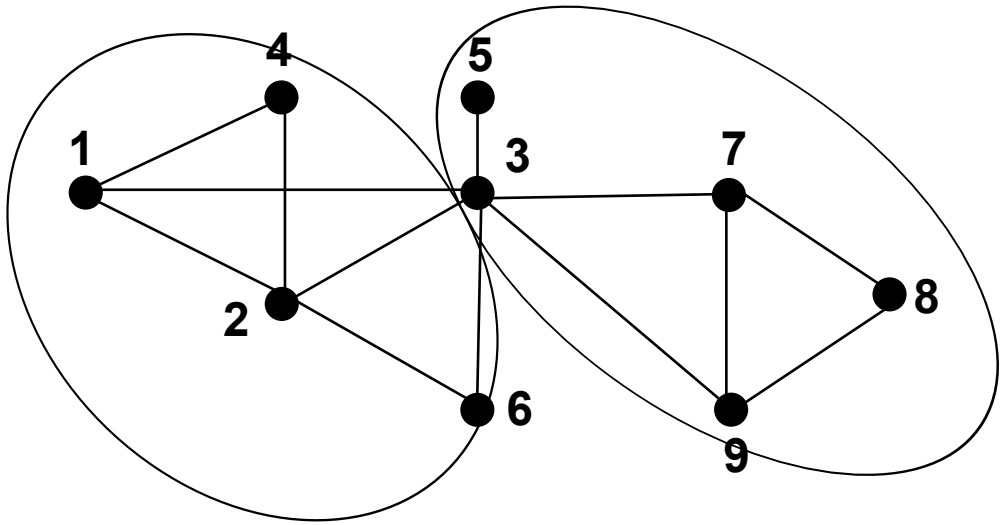
- The maximum is achieved when each sum term is non-negative, i.e., when

$$s_i = \begin{cases} +1 \text{ if } [\mathbf{u}_1]_i > 0 \\ -1 \text{ if } [\mathbf{u}_1]_i < 0 \end{cases} \tag{40}$$

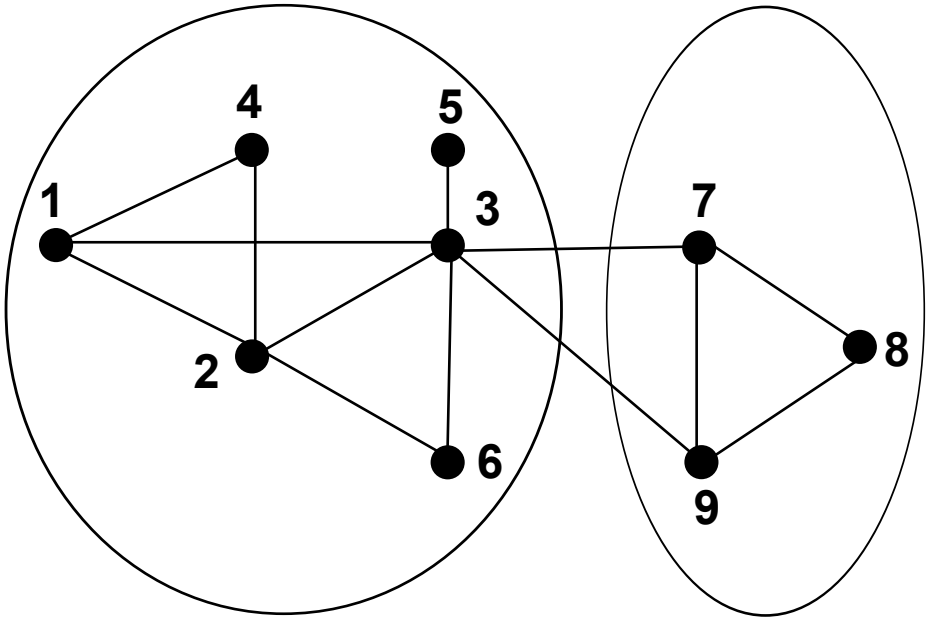
Algorithm

- Community detection algorithm
  1. Calculate the leading eigenvector of the modularity matrix (corresponding to the largest eigenvalue)
  2. We assign nodes to communities according to the signs of vector elements

Example



Example



Complexity of spectral modularity maximization

- One potential problem with the algorithm is that $\mathbf{B}$ is unlike the Laplacian a dense matrix
- Finding the leading eigenvector of a matrix has complexity $O(mn)$, which for a dense matrix is $O(n^3)$
- However, by exploiting some special properties of the modularity matrix this can be achieved in $O(n^2)$
- Overall, the spectral method is as efficient as the simple modularity maximization method

Other modularity maximization methods

- Simulated annealing
- Genetic algorithms
- Greedy algorithm that starts with each node in a single community
- It then aggregate communities in larger communities by picking the aggregation that increases modularity at most

Further heuristics

- Betweenness-based methods: look for links that lie between communities
- Use a betweenness centrality score for links and remove the links with high score
- Hierarchical clustering
- Agglomerative method in which we start with individual nodes and then join them to form groups
- Joining is based on a similarity between nodes

Model-based methods

- A third important method for community detection is based on fitting a probabilistic model of a network
- The idea is based on a probabilistic or *generative* model
- Such models assign a link probability for each pair i and j of nodes in a network
- Generative models are a powerful method to encode specific assumptions of how unknown parameters interact to create links

Generative models

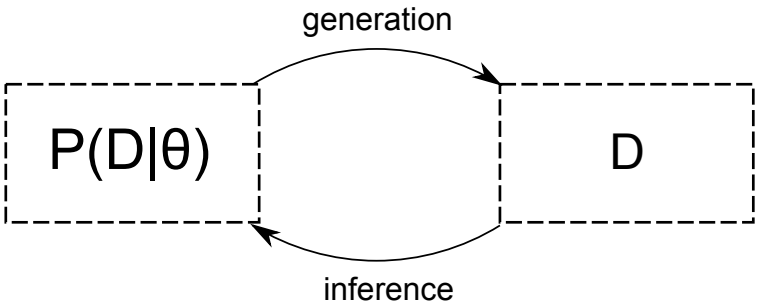
- Generative models have many advantages:
  1. they make our assumptions about the world explicit (rather than encoding them within a procedure or algorithm)
  2. their parameters can be directly interpreted with respect to certain hypothesis about network structure
  3. comparison of different parameterizations is based on likelihood, which is a fundamental principle in probability theory and statistics
  4. they make probabilistic statements about observation (lack-of) specific network features
  5. they allow for prediction of future features based on the past observations

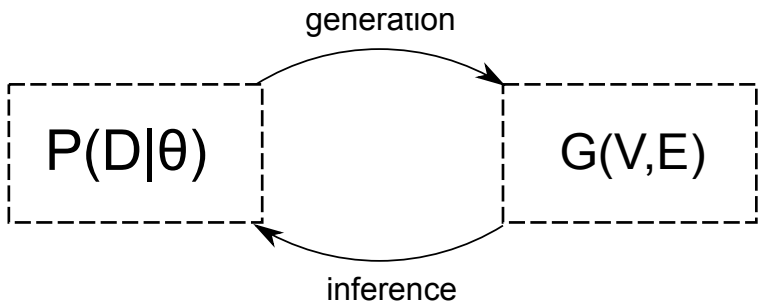
Generative models

- The main disadvantage is that fitting of the models can be more complicated than a heuristic algorithmic approach
- How does a generative network model work?
- It defines a probability distribution over all possible networks $P(G|\theta)$
- θ is a set of parameters that govern the link probabilities under the model
- Given θ we flip coin for each pair of nodes and depending on the result of the coin flip assign or do not assign a link between those nodes
- In that way we generate an *instance* network G from the model

Inference

- (Statistical) *inference* is the reverse of the generation process
- We are given an instance G of the network, e.g. an empirical network
- We want to estimate the model, or more precisely the parameters θ that most likely generated the network





Bernoulli (Erdös-Renyi) model

- The most standard generative model for networks
- We are given n nodes and a link between any two nodes is present with probability p, independently on any other link
- We have $\binom{n}{2}$ potential links in a simple undirected network
- Expected degree of a node is $(n-1)p$ and is same for all nodes
- Degree distribution follows a Poisson distribution

Fitting a Bernoulli model

- We observe a simple undirected network with n nodes and m links
- We are interested in parameter p that most likely generated the network
- p is now an unknown parameter
- Again, we can estimate the parameter with MLE

Fitting a Bernoulli model

- First, we need to write down the likelihood function
- That is the probability of observing the network given parameter p
- We need to iterate through all $\binom{n}{2}$ potential links and write down the probability of observing or otherwise not-observing the link
- As links are independent on each other given the parameter p the final probability is the product of single probabilities

Fitting a Bernoulli model

$$
\begin{aligned}
P(G|p) &= \underbrace{p \cdot p \ldots p}_{m} \underbrace{\cdot (1-p) \cdot (1-p) \ldots (1-p)}_{\binom{n}{2}-m} \\
&= p^m (1-p)^{\frac{n(n-1)}{2}-m} \qquad (41)
\end{aligned}
$$

Log-likelihood

$$
\mathcal{L}(p) = m ln(p) + (\frac{n(n-1)}{2} - m) ln(1-p) \qquad (42)
$$

MLE for a Bernoulli model

- Now, we are interested in p that most likely generated the data
- The data are most likely to have been generated by the model with p that maximizes the log-likelihood function
- Setting $\frac{d\mathcal{L}}{dp} = 0$ and solving for p we obtain the *maximum likelihood estimate*

MLE

$$\frac{d\mathcal{L}}{dp} = \frac{m}{p} - \frac{\frac{n(n-1)}{2} - m}{1-p} = 0 \tag{43}$$

$$p = \frac{2m}{n(n-1)} \tag{44}$$

Erdös-Renyi mixture models

- It is also known as *latent block models*
- Also *stochastic block models* (SBM)
- Social science terminology
- Nodes are of different types, e.g. there are k different types
- The probability of a link varies depending on node types

Stochastic block models

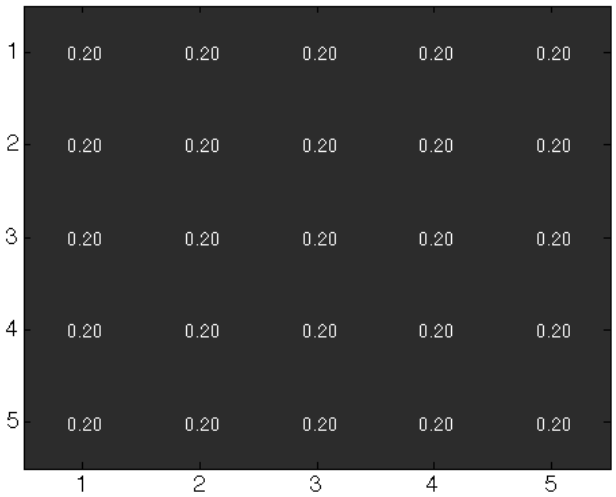
- In its most simple form an SBM is defined by
  1. Number of nodes n
  2. Number of types k
  3. $n \times 1$ vector $\mathbf{s}$ of node type indices
  4. $k \times k$ stochastic block matrix $\mathbf{M}$ where M_{uv} gives the probability that a node of type u is connected to a node of type v

Stochastic block models

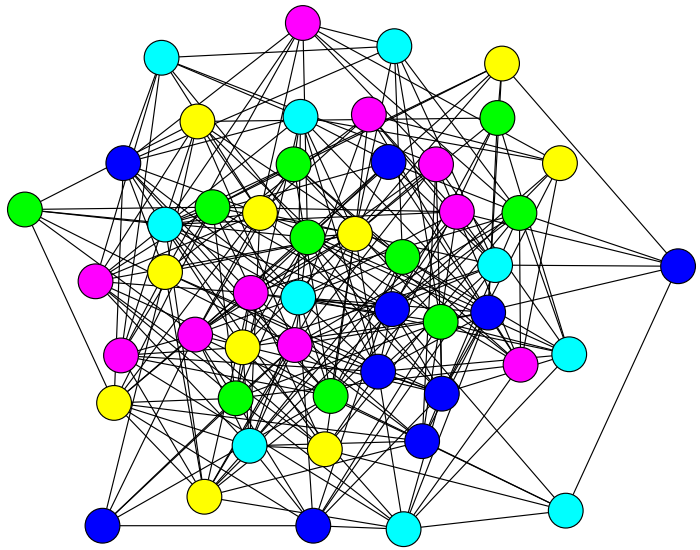
- Given a choice for n, k, $\mathbf{s}$ and $\mathbf{M}$ we can draw a network instance from the model
- We iterate through all $\binom{n}{2}$ potential links and flip a coin for each link
- Each link is a Bernoulli r.v. with parameter $p = M_{uv}$, where u and v are types of two nodes in question
- Links are independent but not identically distributed
- However, they are i.i.d. for a given pair of types u and v
- Note that (for undirected case) we need to specify $\binom{k+1}{2}$ probabilities in $\mathbf{M}$

Stochastic block models

- For $M_{uv} = p$ constant for all pairs of types u and v we get a Bernoulli random graph



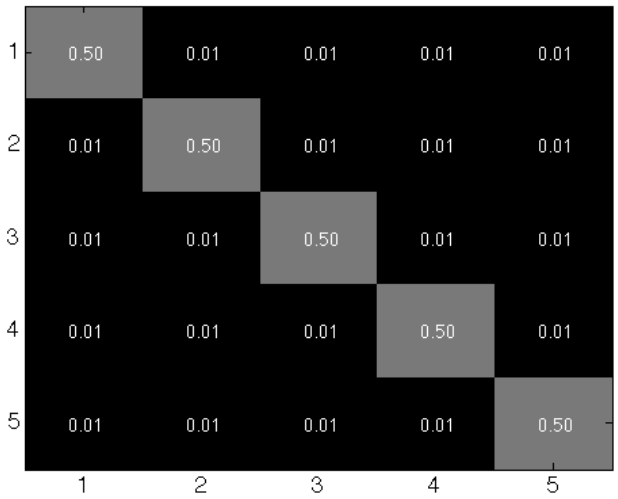
stochastic block matrix



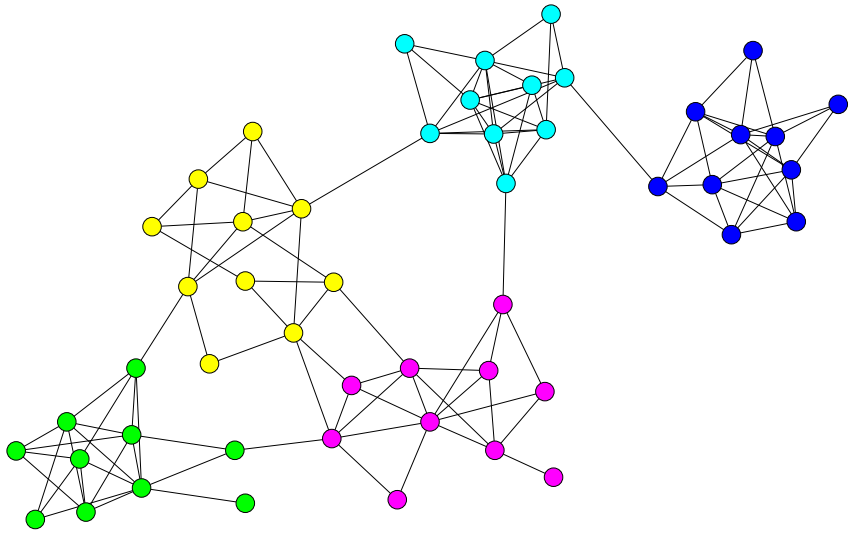
random graph

Stochastic block models

- For $M_{uu} > M_{uv}, u \neq v$ diagonal elements greater than off diagonal elements we get assortative mixing



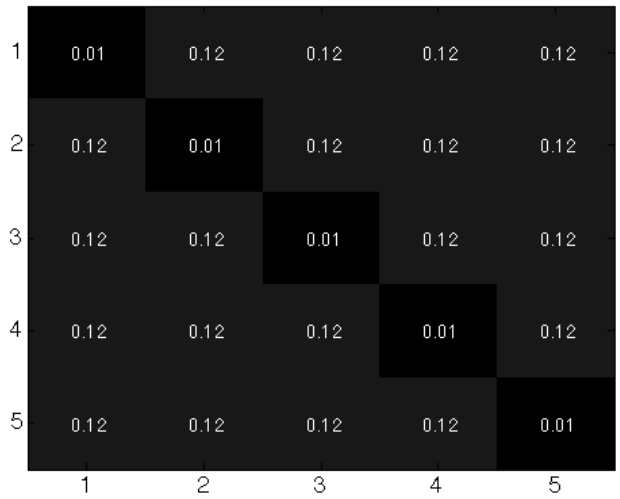
stochastic block matrix



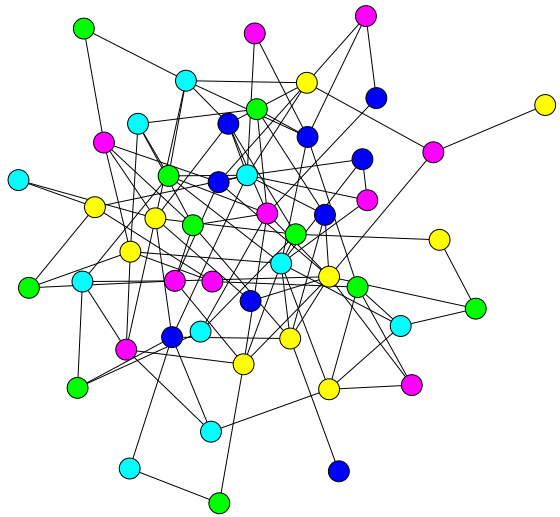
assortative communities

Stochastic block models

- For $M_{uu} < M_{uv}, u \neq v$ diagonal elements smaller than off diagonal elements we get disassortative mixing



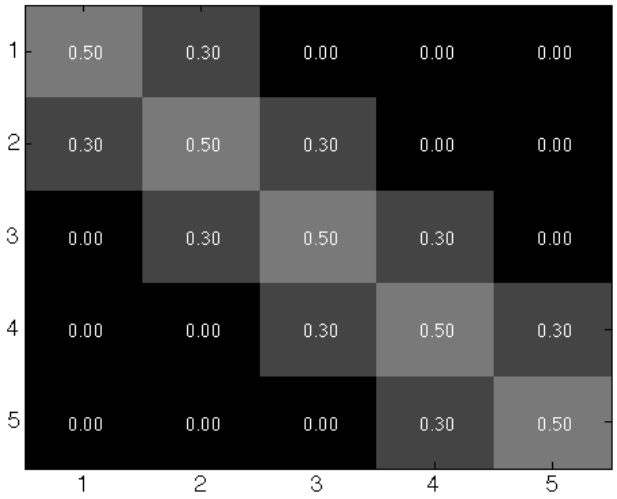
stochastic block matrix



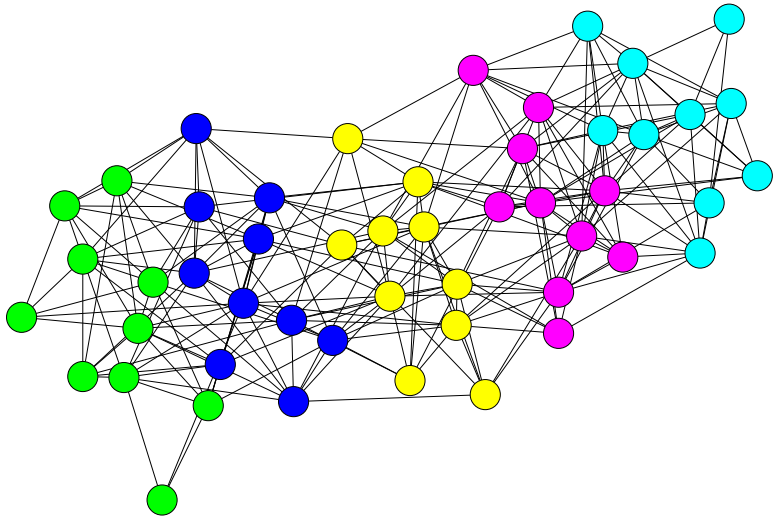
disassortative communities

Stochastic block models

- In ordered networks individuals connect to other individuals according to a latent sequence



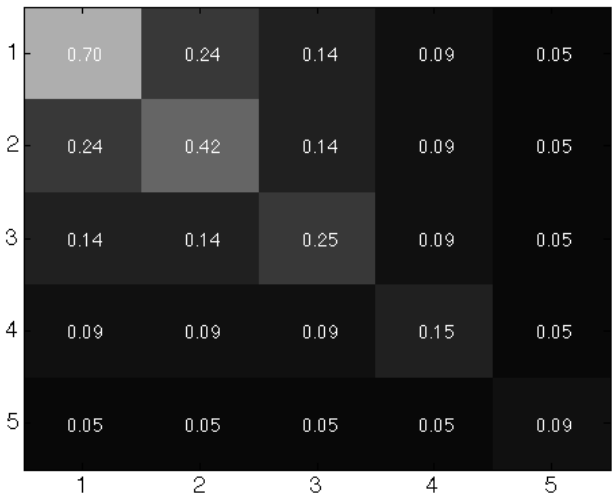
stochastic block matrix



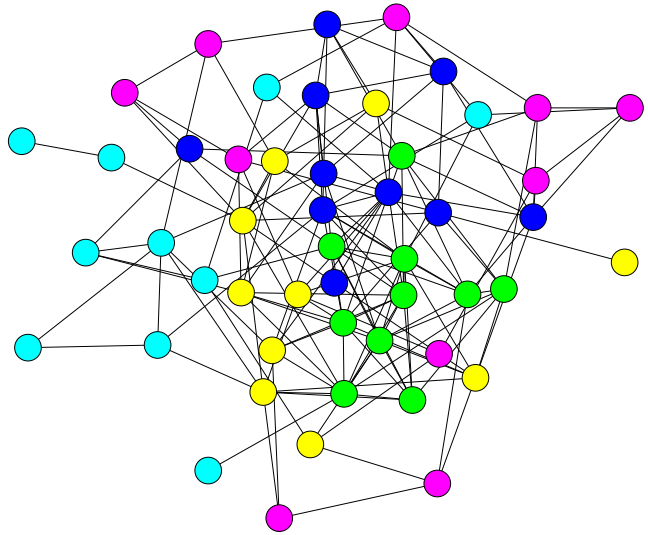
ordered communities

Stochastic block models

- Ordered communities but the density of connections decreases with a community index



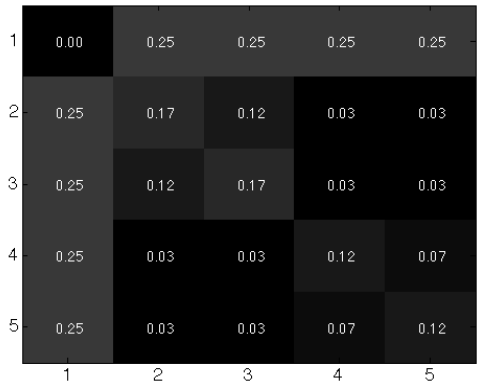
stochastic block matrix



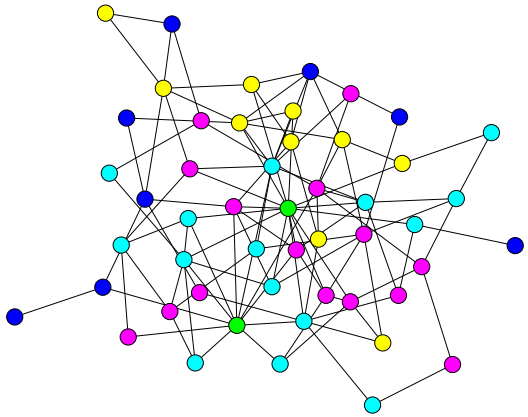
core-periphery structure

Stochastic block models

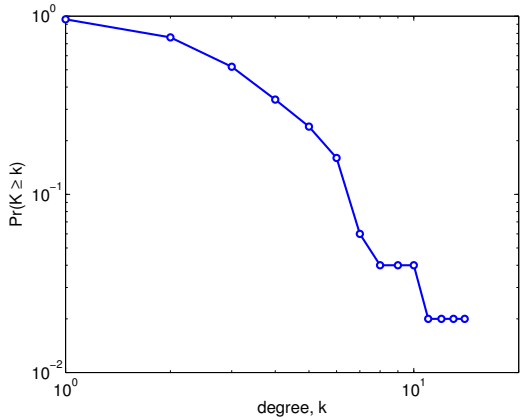
- Power-law degree distribution is achieved by a small community that tends to connect to other communities



stochastic block matrix



heterogeneous degrees



degree distribution

Stochastic block models

- SBM can naturally handle directed networks
- We relax the previous assumption that $\mathbf{M}$ is symmetric
- In this way the probability of a link running in one direction is different of the probability that a link runs in the opposite direction

SBM: Inference

- Given a choice of k and an observed network G we can use the SBM to infer the model parameters
- I.e. we infer the vector of group assignments $\mathbf{s}$ and the stochastic block matrix $\mathbf{M}$
- The simplest method is to use MLE
- We aim to select $\mathbf{s}$ and $\mathbf{M}$ that maximize the likelihood of observing the data

Fitting an SBM

$$
\begin{aligned}
P(G|\mathbf{M},\mathbf{s}) &= \prod_{i,j} P(i,j|\mathbf{M},\mathbf{s}) \\
&= \prod_{(i,j)\in E} P(i,j|\mathbf{M},\mathbf{s}) \prod_{(i,j)\notin E} (1 - P(i,j|\mathbf{M},\mathbf{s})) \qquad (45)
\end{aligned}
$$

- For a simple undirected graph the product contains $\binom{n}{2}$ terms

Fitting an SBM

- For each of $k \times k$ blocks with $u \neq v$ holds
- The number of potential links is given by $n_{uv} = n_u n_v$ where n_u and n_v are the number of nodes in block u and v respectively
- M_{uv} is the probability of a link between nodes from block u and v
- The number of links is binomially distributed r.v. with parameters n_{uv} and M_{uv}
- If the number of observed links is m_{uv} then the MLE for that block is given by

MLE for a single block ($u \neq v$)

$$\hat{M}_{uv} = \frac{m_{uv}}{n_{uv}} \tag{46}$$

Fitting an SBM

- For each of k blocks with $u = v$ holds
- The number of potential links for a simple undirected network is given by $n_{uu} = \binom{n_u}{2}$
- Keeping this in mind the MLE for diagonal blocks is again given by

MLE for a single block

$$\hat{M}_{uv} = \frac{m_{uv}}{n_{uv}} \tag{47}$$

Fitting an SBM

- Now we substitute MLEs for M_{uv} into the likelihood function for the complete network

Likelihood for an SBM

$$
\begin{aligned}
P(G|\mathbf{M},\mathbf{s}) &= \prod_{(i,j)\in E} M_{s_u,s_v} \prod_{(i,j)\notin E} (1 - M_{s_u,s_v}) \\
&= \prod_{u,v} M_{uv}^{m_{uv}} (1 - M_{uv})^{(n_{uv}-m_{uv})} \\
&= \prod_{u,v} (\frac{m_{uv}}{n_{uv}})^{m_{uv}} (1 - \frac{m_{uv}}{n_{uv}})^{(n_{uv}-m_{uv})} \qquad (48)
\end{aligned}
$$

Fitting an SBM

- Log-likelihood is then given by:

Likelihood for an SBM

$$\begin{aligned}\mathcal{L} &= \sum_{u,v}(m_{uv}ln(\frac{m_{uv}}{n_{uv}}) + (n_{uv} - m_{uv})ln(1 - \frac{m_{uv}}{n_{uv}})) \\ &= \sum_{u,v}(m_{uv}ln(m_{uv}) + (n_{uv} - m_{uv})ln(n_{uv} - m_{uv}) - n_{uv}ln(n_{uv})) \end{aligned} \quad (49)$$

MLE for an SBM

- Now, we are interested in $\mathbf{s}$ that maximizes $\mathscr{L}$
- As before, one possibility is to maximize $\mathscr{L}$ with Kernighan-Lin algorithm
- Another possibility would be sampling (Markov Chain Monte Carlo)
- E.g. random selection of nodes to move from one group to another
- Rejection of moves that decrease $\mathscr{L}$
- After sufficiently large number of movements we find the maximum of $\mathscr{L}$

MCMC: Simulated annealing

- MCMC methods originate from Metropolis procedure
- Originally, efficient simulation of a collection of atoms in equilibrium at a specific temperature
- In each step of the algorithm you give a small displacement to an atom
- This results in the change of the energy: ΔE
- If $\Delta E \leq 0$ then we accept the movement
- Otherwise we accept it with the probability $P(\Delta E) = e^{\frac{-\Delta E}{kT}}$

MCMC: Simulated annealing

- k Boltzmann constant and T is the temperature
- By repeating the step many times we simulate the thermal motion of atoms
- The system evolves into the state of the minimal energy (Boltzmann distribution)
- Using the cost function instead of energy we generate a population of configurations for a given optimization problem
- E.g. the cost function in our case would be $-Q$ or $-\mathcal{L}$

MCMC: Simulated annealing

- T is a control parameter
- At high temperatures you observe gross features of the system
- At lower temperatures we develop fine details
- Simulated annealing does exactly this: it iterates over the temperatures
- Starts with a higher temperature, and after reaching a steady state lowers the temperature and repeats the process

MCMC: Simulated annealing

- Atom movements in case of community detection are movements of nodes from one group into other
- Thus, you start with a random division and calculate $-Q$ or $-\mathcal{L}$
- You move at random a single node from one group into another
- Calculate the difference in the objective function and accept if the difference is negative
- Otherwise accept with the probability from above
- Then lower the temperature and repeat: http://www.nature.com/nature/journal/v433/n7028/full/nature03288.html

Simple example

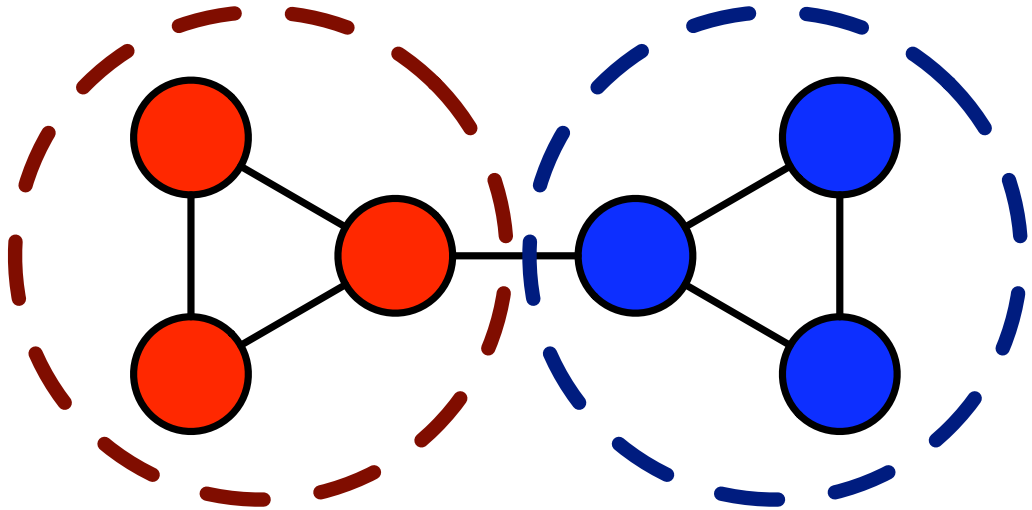


Figure: “Good” likelihood

Simple example

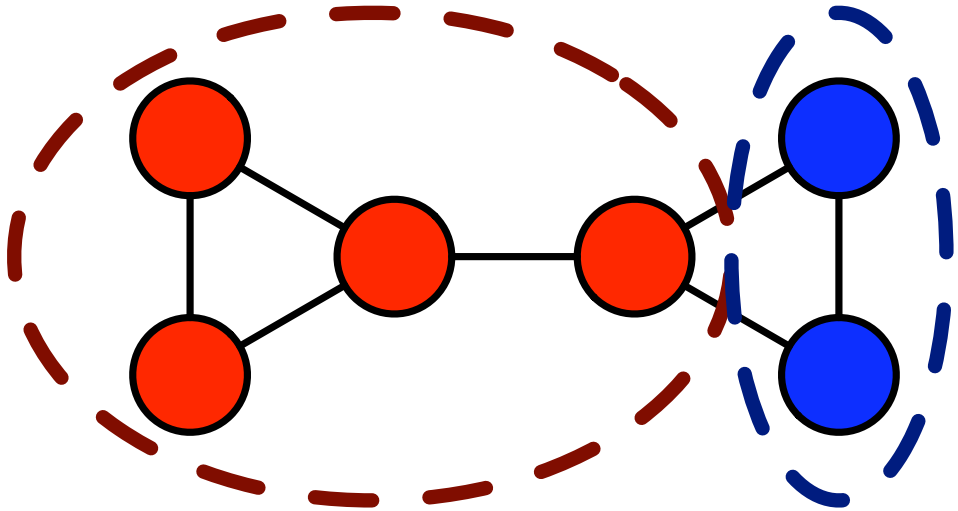


Figure: “Bad” likelihood

How many groups?

- What happens if we do not fix the number of groups k?
- How does likelihood behave if we increase or decrease k?
- The larger k we take the more parameters we have in $\mathbf{M}$
- More parameters mean a better likelihood and a better fit
- However, there exists a danger of *overfitting*
- I.e. we fit very well to a given instance of network but are not able to generalize to other instances

How many groups?

- In extreme case if $k = n$, every node is in group for itself
- The number of potential links to any other group is always 1
- Now, MLE of a binomial for each block equals $\frac{m_{uv}}{n_{uv}}$
- We have $n_{uv} = 1, \forall u \neq v$
- We also have $m_{uv} = 0$ if there is no link between the groups and $m_{uv} = 1$ if there is a link

How many groups?

- MLE for M_{uv} is either 0 (no link) or 1 (link)
- In fact we obtain the adjacency matrix $\rightarrow$ the model has memorized the data exactly and can not generalize
- The likelihood is in this case maximal and equals 1
- In such situations we typically penalize the increased number of parameters
- Regularization, complexity control, penalized likelihood, etc.

An alternative SBM

- We model the number of links between any pair of nodes as a Poisson r.v. with parameter $\lambda = M_{uv}$, where u and v are types of two nodes in question
- This approximation works quite well if M_{uv} are small, which they are for sparse networks
- This approximation simplifies the calculation of likelihood

Likelihood for a Poisson SBM

$$\mathcal{L}(G|\mathbf{s}) = \sum_{u,v} m_{uv} ln \frac{m_{uv}}{n_{uv}} \tag{50}$$

Poisson SBM

- By adding and dividing with a number of nodes and links and by dropping irrelevant constants we get the following equation:

Likelihood for a Poisson SBM

$$\mathcal{L}(G|\mathbf{s}) = \sum_{u,v} \frac{m_{uv}}{2m} ln \frac{m_{uv}/2m}{n_{uv}/n^2} \tag{51}$$

Poisson SBM

- Now imagine that we choose a link uniformly at random from the network
- What is the probability that one end of the link will be of type u and another of type v?
- $P(r,s) = \frac{m_{uv}}{2m}$
- Now imagine that we select two nodes uniformly at random from the network (and create a link between them)
- What is the probability that node will be of type u and another of type v?
- $Q(r,s) = \frac{n_{uv}}{n^2}$

Poisson SBM

- Now we can write the likelihood as:

Likelihood for a Poisson SBM

$$\mathcal{L}(G|\mathbf{s}) = \sum_{u,v} P(u,v) ln \frac{P(u,v)}{Q(u,v)} \tag{52}$$

Information-theoretic perspective on SBM

- This is the Kullback-Leibler divergence between $P(u, v)$ and $Q(u, v)$
- It measures the expected number of extra bits required to encode u and v if we use Q (null-model) instead of a true model P
- Intuitively, it measures how far is P from Q
- The most likely assignments under the Poisson SBM are those assignments that require the most information to describe the network starting from a model that does not have group structure
- In this case random graph model, or Erdös-Renyi model

Difference between observation and expectation

- We already have had this before!
- We constructed an objective function which measures the difference between the observed quantity and the expected value of the same quantity under an appropriate null model
- Modularity!

$$Q = \frac{1}{2m} \sum_{ij} (A_{ij} - \frac{k_i k_j}{2m}) \delta(c_i, c_j) \tag{53}$$

Difference between observation and expectation

- However, modularity incorporated degree sequence into the null model
- A random graph has Poisson distribution of degrees
- We typically observe power-law distributions
- This leads to Degree-Corrected Stochastic Block Model

Degree-corrected SBM

- We model the number of links between any pair of nodes as a Poisson r.v. with parameter $\lambda = \theta_i \theta_j M_{uv}$, where u and v are types of two nodes in question, and θ_i is the probability that a link to a particular group lands on node i
- Likelihood is then given by:

Likelihood for a Poisson SBM

$$\mathcal{L}(G|\mathbf{s}) = \sum_{u,v} m_{uv} ln \frac{m_{uv}}{\kappa_u \kappa_v} \tag{54}$$

- κ_u is the sum of degrees in the group u

Degree-corrected SBM

- Again, by adding and dividing with a number of nodes and links and by dropping irrelevant constants we get the following equation:

Likelihood for a degree-corrected SBM

$$\mathcal{L}(G|\mathbf{s}) = \sum_{u,v} \frac{m_{uv}}{2m} ln \frac{m_{uv}/2m}{(\kappa_u/2m)(\kappa_v/2m)} \tag{55}$$

Information-theoretic perspective on degree-corrected SBM

- Again, this is the Kullback-Leibler divergence between the observed model and the null-model
- However, in this case the null-model is a random graph but with correct degree sequence
- The most likely assignments under the degree-corrected SBM are those assignments that require the most information to describe the network starting from a model that does not have group structure but have the same degree sequence
- This is exactly the modularity
- Thus, there is a strong analogy between fitting degree-corrected SBM and maximizing modularity