Community detection

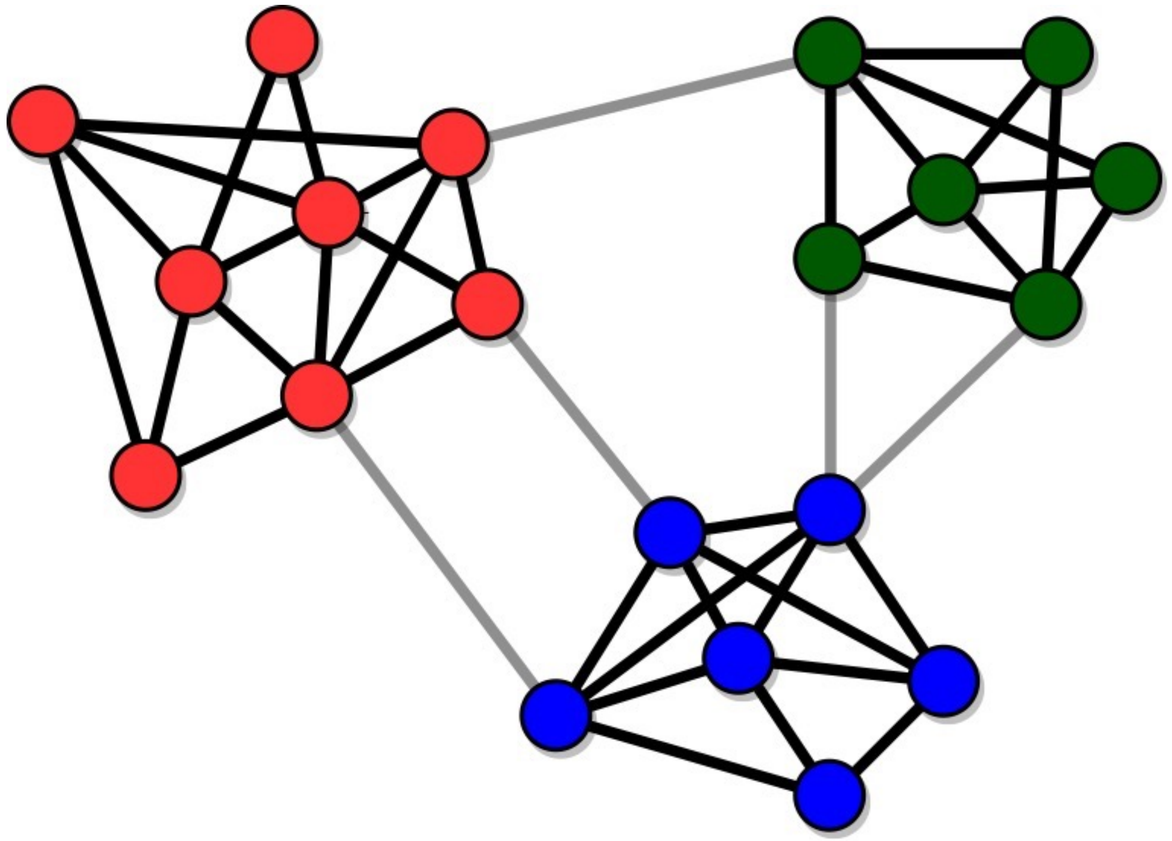
Introduction to Network Science

Instructor: Michele Starnini — https://github.com/chatox/networks-science-course

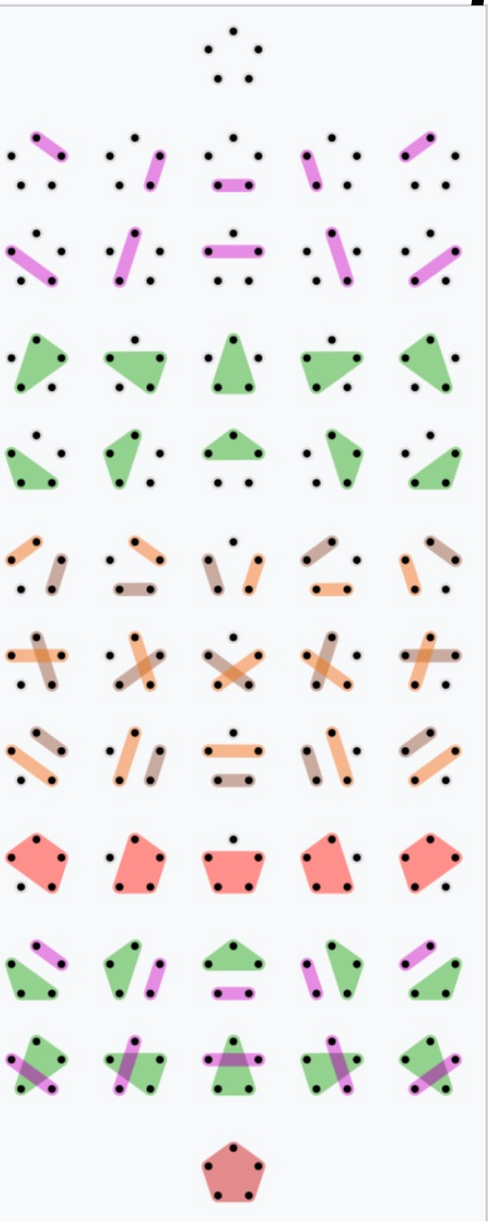
upf. Universitat Pompeu Fabra Barcelona

Community detection

- Many different methods
- We discuss these techniques:
  - **Bridge removal**
  - **Modularity optimization**
  - **Label propagation**
  - **Stochastic block modeling**



Possible Partitions

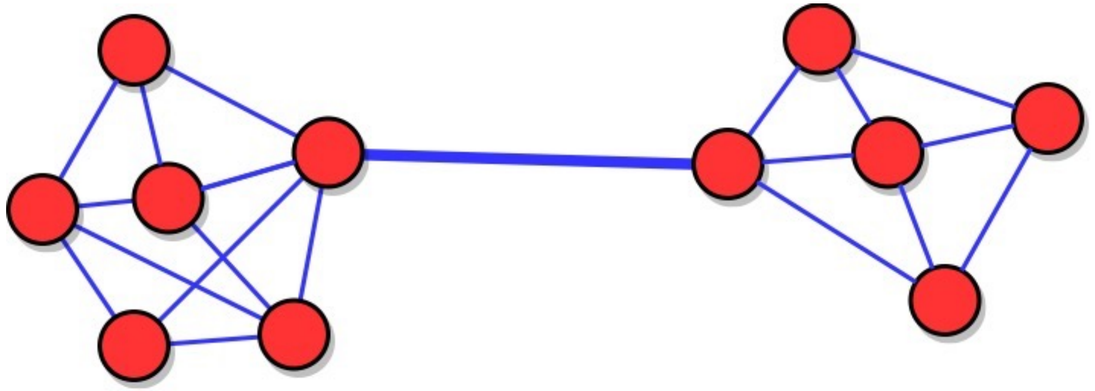


- The number of partitions of n objects is the **Bell number B_n**
- The Bell number **grows faster than exponentially with *n***
- **Conclusion:** it makes no sense to look for interesting community structures by exploring the whole space of partitions! A smart exploration of the partition space must be performed.

n	B_n
1	1
2	2
3	5
4	15
5	52
6	203
7	877
8	4140
9	21147
10	115975
11	678570
12	4213597
13	27644437
14	190899322
15	1382958545

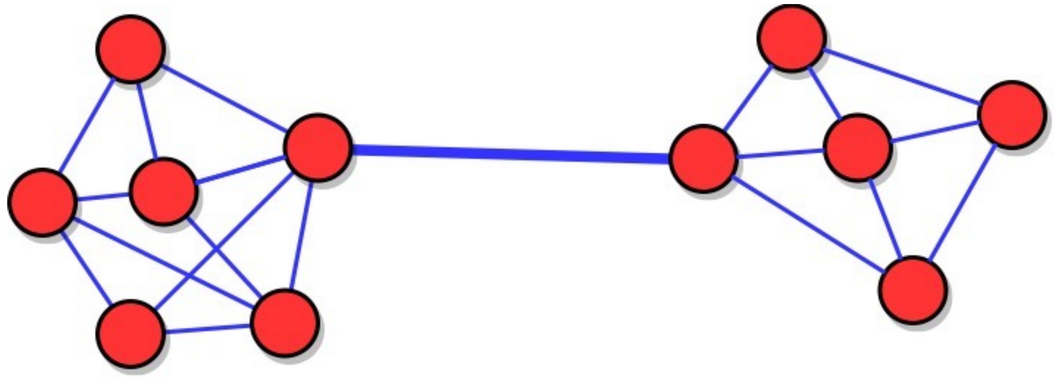
Bridge removal

- **Divisive method:** one removes the links that connect the clusters to each other (**bridges**), until they are isolated
- The problem would then be solved by finding the connected components of the resulting disconnected graph (trivial!)
- Bridges are identified via measures that take larger (smaller) values on those links than on internal links



Girvan-Newman algorithm

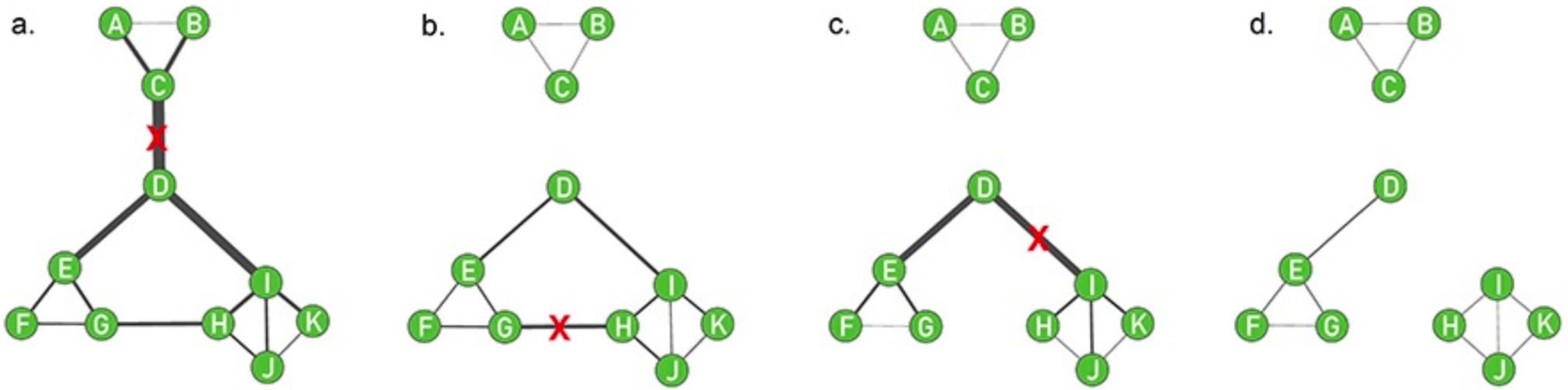
- Measure for bridge identification: **link betweenness**
- Bridges are expected to have large betweenness values because shortest paths between nodes in different communities run through bridges
- Instead, internal links are expected to have comparatively lower betweenness values, because there are many alternative routes going from one node of the community to another, due to the high density of links inside the cluster(s)



Girvan-Newman algorithm

Repeat:

- **Compute edge betweenness**
- **Remove edge with larger betweenness**



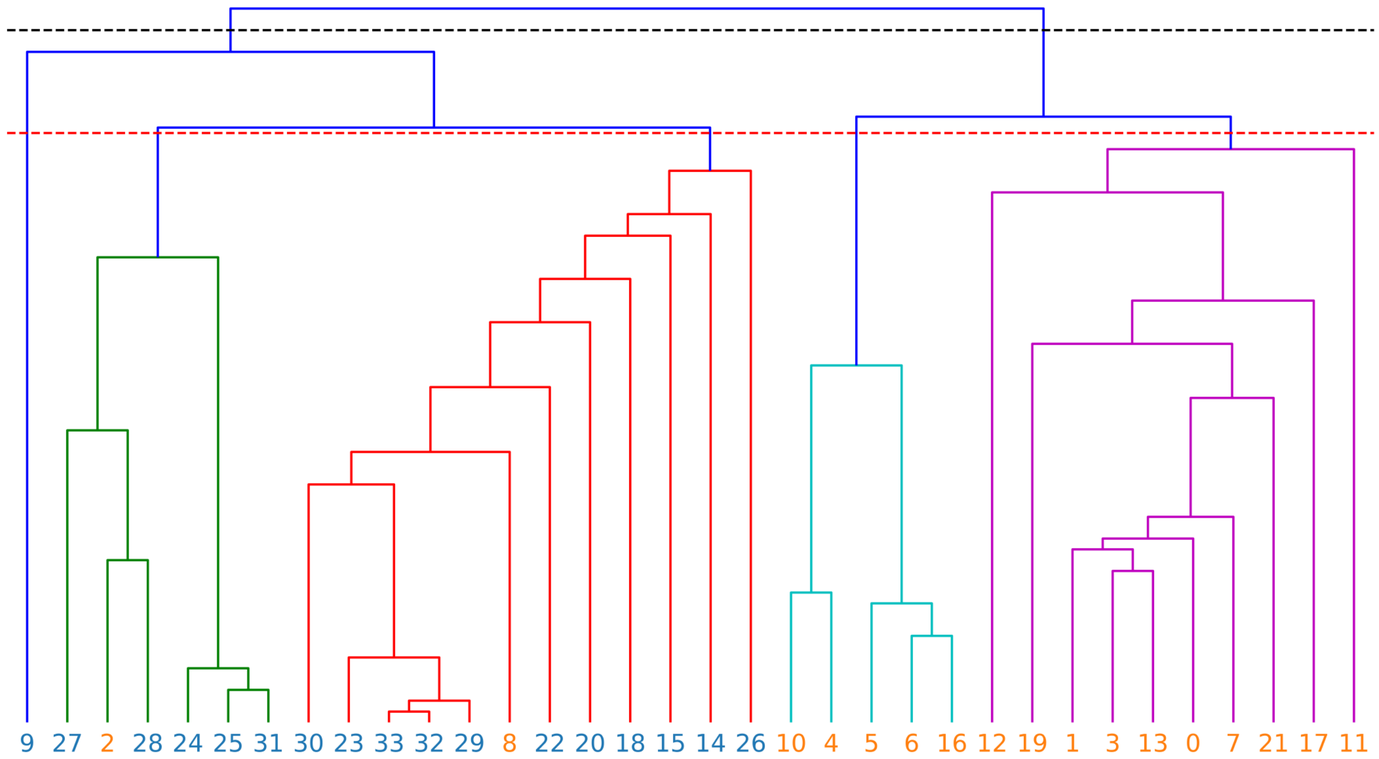
Girvan-Newman algorithm

- The recalculation of the betweenness at each iteration is necessary, but it makes the algorithm slow
- On networks with strong community structure, which quickly break into disconnected communities, the recalculation step needs to be performed only within the connected component including the last removed link, as the betweenness of all other links remains the same
- This can reduce the complexity of the procedure

Girvan-Newman algorithm

- The Girvan-Newman algorithm is a **divisive hierarchical clustering** method, as it delivers hierarchical partitions by breaking clusters until only **singletons** (isolated nodes) remain

- Output: **dendrogram!**



Girvan-Newman algorithm: limits

- It is quite slow, it is not practical for large networks with, say, more than 10,000 nodes. The bottleneck is the recalculation of the link betweenness

- Faster variants have been proposed, for instance computing approximations of the link betweenness scores by using only a sample of randomly selected pairs of nodes, or adopting alternative measures to identify bridges, which are quicker to compute

- The method delivers a full hierarchy of *N* partitions: which ones are meaningful and how can they be selected?

```
# returns a list of hierarchical partitions
partitions = nx.community.girvan_newman(G)
```

Quality functions

- **Question:** how can we say how good a partition is?
- **Answer:** quality functions!
- **Issues:**
  - The internal link density of the clusters is not enough to assess their community quality. **Random networks have no communities**, so any subnetwork of a random network is not eligible as community no matter how dense the subnetwork is internally
  - It is necessary to distinguish actual communities, where there are high concentrations of links because of specific features of their nodes (e.g., similarity), from **pseudo-communities**, where high concentrations of links are produced by chance in the construction process of the network

Modularity

- **Principle:** evaluating communities with respect to a **random baseline**
- **Baseline:** randomized versions of the original network, preserving its degree sequence
- For each community of a partition, modularity computes the **difference** between the number of internal links in the community and the expected value of this number in the set of randomized networks
- If the network is random (e.g., Erdős–Rényi), the modularity of any partition is supposed to be low, because the number of internal links of any cluster of the partition should be close to the expected value in the randomized networks
- If the number of links within the clusters is much larger than its expected random value, it is unlikely for such a concentration of internal links to be the result of a random process, and modularity can reach high values

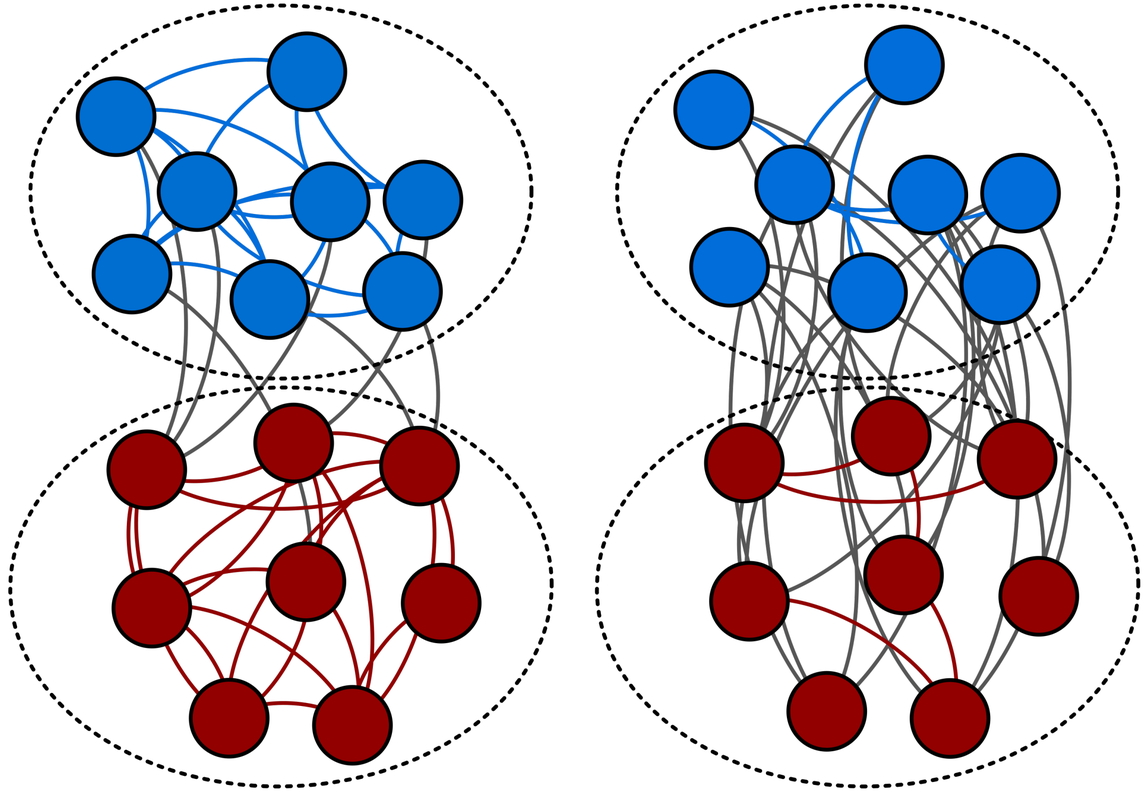
Modularity

$$Q = \frac{1}{L} \sum_{C} \left(L_C - \frac{k_C^2}{4L} \right)$$

- L = number of links in the network
- L_C = number of internal links in community C
- k_C = degree of community C
- $k^2_C/4L$ = expected number of internal links in community C

Modularity: null model

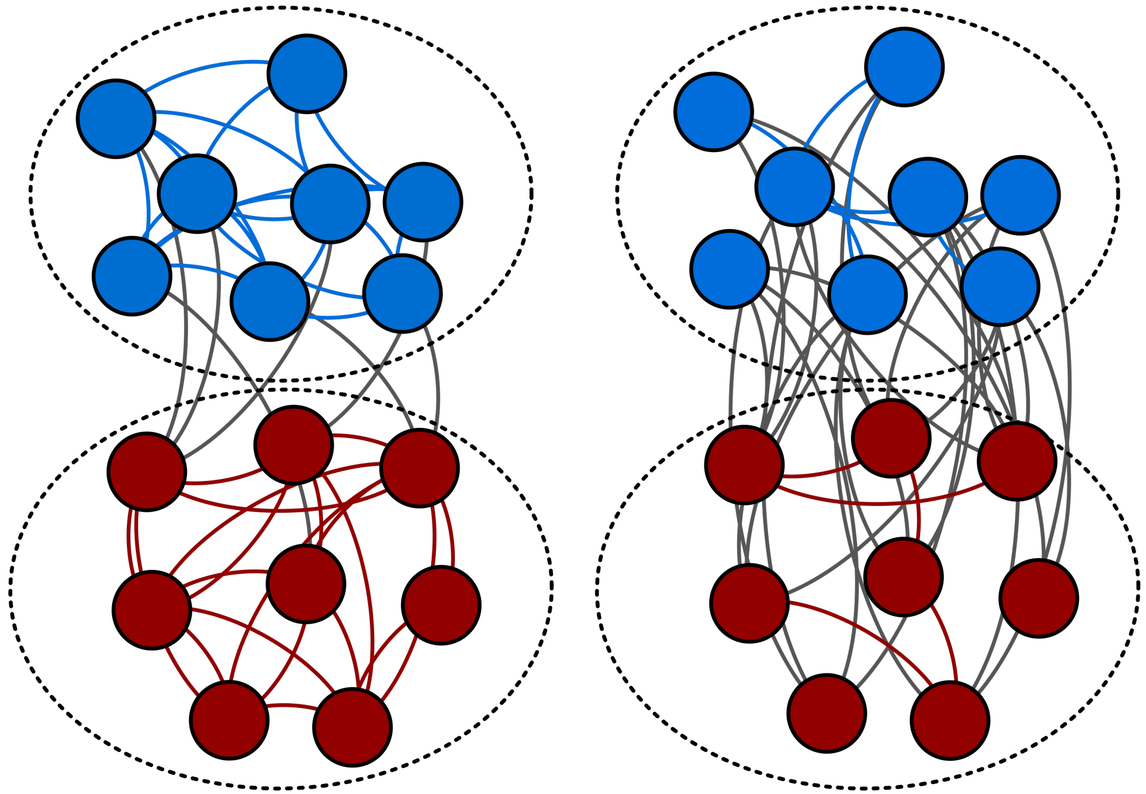
- Let's explain the origin of $k^2_C / 4L$
- Random links are formed by matching pairs of **stubs** (half-links) chosen at random
- The total number of stubs attached to C is k_C
- The probability to select one of those stubs at random is $k_C/2L$ because $2L$ is the total number of stubs of the network (each link yields two stubs)



Original network **Randomized network**

Modularity: null model

- For a random link to connect two nodes in the same cluster C, two stubs must be selected from C
- The probability to pick two stubs from C at random is (roughly) the product of the probabilities of selecting each one: $p_C = (k_C/2L) \cdot (k_C/2L) = k^2_C / 4L^2$
- Since there are L links in the network, and each has a probability p_C to end up within C, the expected number of internal links in C is $k^2_C / 4L$

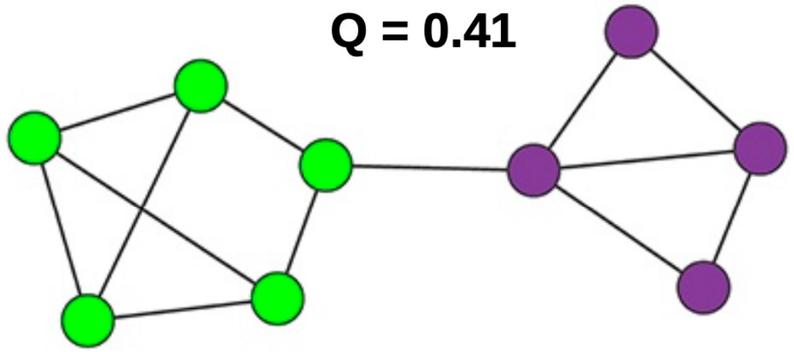


Original network **Randomized network**

Modularity examples

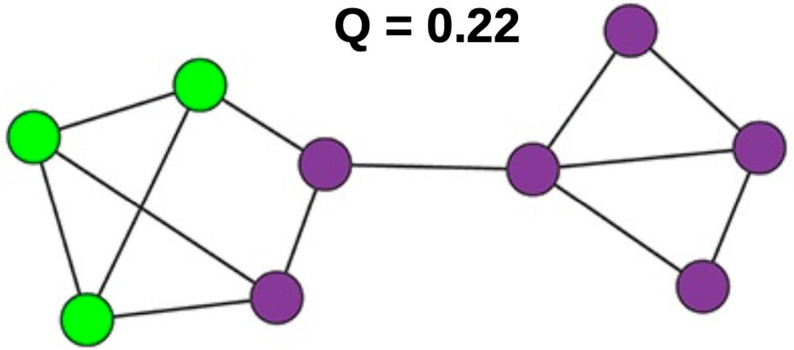
a. OPTIMAL PARTITION

Q = 0.41



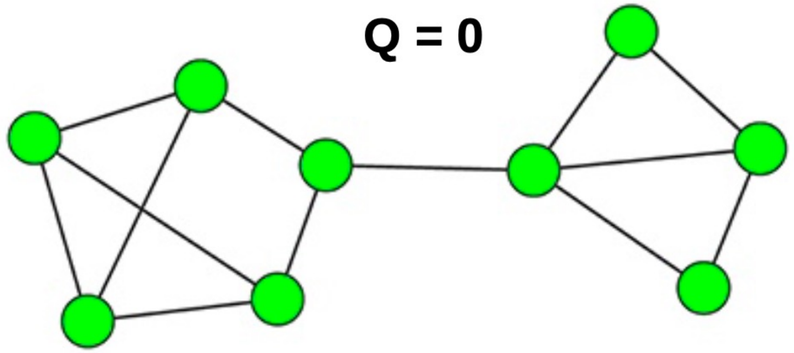
b. SUBOPTIMAL PARTITION

Q = 0.22



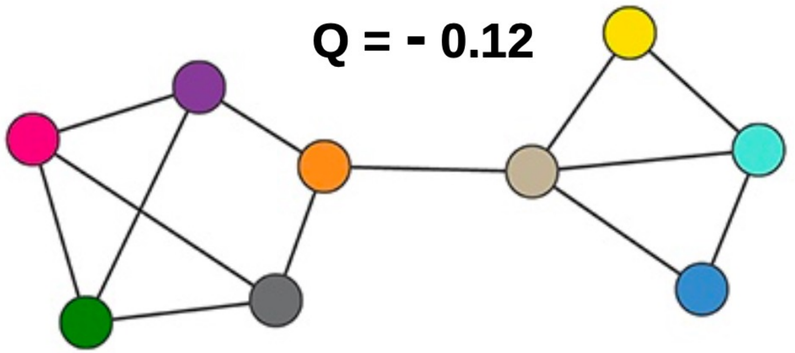
c. SINGLE COMMUNITY

Q = 0



d. NEGATIVE MODULARITY

Q = - 0.12



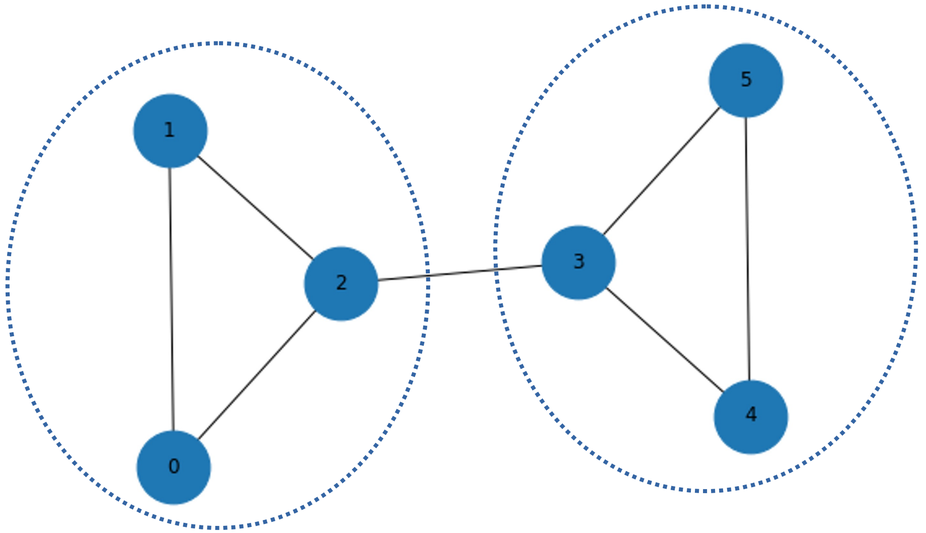
© 2020 F. M

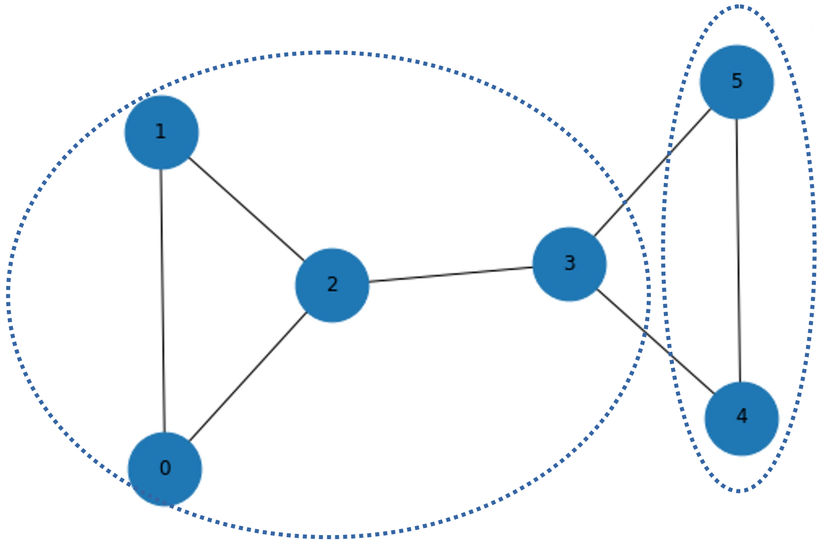
$$Q = \frac{1}{L}\sum_{C}\left(L_C - \frac{k_C^2}{4L}\right)$$

Exercise

- What is the modularity of the partition {0, 1, 2}, {3, 4, 5}?

- What is the modularity of the partition {0, 1, 2, 3}, {4, 5}?





$$Q = \frac{1}{L} \sum_{C} \left(L_C - \frac{k_C^2}{4L} \right)$$

Pin board: https://upfbarcelona.padlet.org/chato/lak2lnp9m3jc1naj

© 2020 F. Mancini, S. Fortunato & C.A. Davis. Cambridge University Press guide to Network Science

Modularity: features

- $Q < 1$ for every partition of any network
- $Q = 0$ for the partition in which the whole graph is one community
- Q can be negative (e.g., partition in N groups of one node each)
- For most networks, Q has a non-trivial maximum between 0 and 1

```
# returns the modularity of the input partition
modularity = nx.community.quality.modularity(G,partition)
```

Modularity: extensions

Directed networks

$$Q_d = \frac{1}{L} \sum_C \left(L_C - \frac{k_C^{in} k_C^{out}}{L} \right)$$

- L = number of links in the network
- L_C = number of internal links of community C
- k^{in}_C = total in-degree of nodes in community C
- k^{out}_C = total out-degree of nodes in community C

Modularity: extensions

Weighted networks

$$Q_w = \frac{1}{W} \sum_C \left(W_C - \frac{s_C^2}{4W} \right)$$

- W = total weight of network links
- W_C = total weight of internal links in community C
- s_C = total strength of nodes in community C, i.e., sum of the strengths of the nodes in C

Modularity: extensions

Directed and weighted networks

$$Q_{dw} = \frac{1}{W} \sum_C \left(W_C - \frac{s_C^{in} s_C^{out}}{W} \right)$$

- W = total weight of network links
- W_C = total weight of internal links in community C
- s^{in}_C = total in-strength of nodes in community C
- s^{out}_C = total out-strength of nodes in community C

Modularity optimization

$$Q = \frac{1}{L} \sum_{C} \left(L_C - \frac{k_C^2}{4L} \right)$$

- Modularity was introduced to provide a criterion to choose the best partition out of those found via the Girvan-Newman algorithm
- But if modularity is reliable, why not maximize it directly?
- **Modularity optimization:** finding the maximum of Q in the space of all possible partitions of the network into communities
- **Hard problem!**

Newman’s greedy algorithm

Procedure:

- Start: partition with one node in each community
- Merge the pair of groups of nodes that yields the highest increase (lowest decrease) of Q
- Continue until all nodes are in the same community
- Pick the partition with largest modularity

```
# returns the maximum modularity partition
partition = nx.community.greedy_modularity_communities(G)
```

Newman’s greedy algorithm: limits

- The method is greedy, in that it tries to maximize the modularity at each step. As such, it is likely to get stuck on solutions with sub-optimal modularity
- It tends to generate unbalanced partitions, with some clusters much larger than the others. Because of that, the method is not very fast
- Merging groups of similar size or merging more than two groups at a time mitigates the problem

Louvain algorithm

Repeat:

- Phase 1: Modularity is optimized by allowing only local changes to node-communities memberships
- Phase 2: The identified communities are aggregated into super-nodes to build a new network

Louvain algorithm

Phase 1: Modularity optimization

Put each node into a distinct community (one node per community)

For each node i, the algorithm performs two calculations:

- Compute the modularity delta (ΔQ) when putting node i into the community of some neighbor j
- Move i to a community of node j that yields the largest gain in ΔQ

 Phase 1 runs until no movement yields a gain

This first phase stops when a local maxima of the modularity is attained, i.e., when no individual node move can improve the modularity. Note that the output of the algorithm depends on the order in which the nodes are considered. Research indicates that the ordering of the nodes does not have a significant influence on the overall modularity that is obtained

Louvain algorithm

Phase 2: Community aggregation

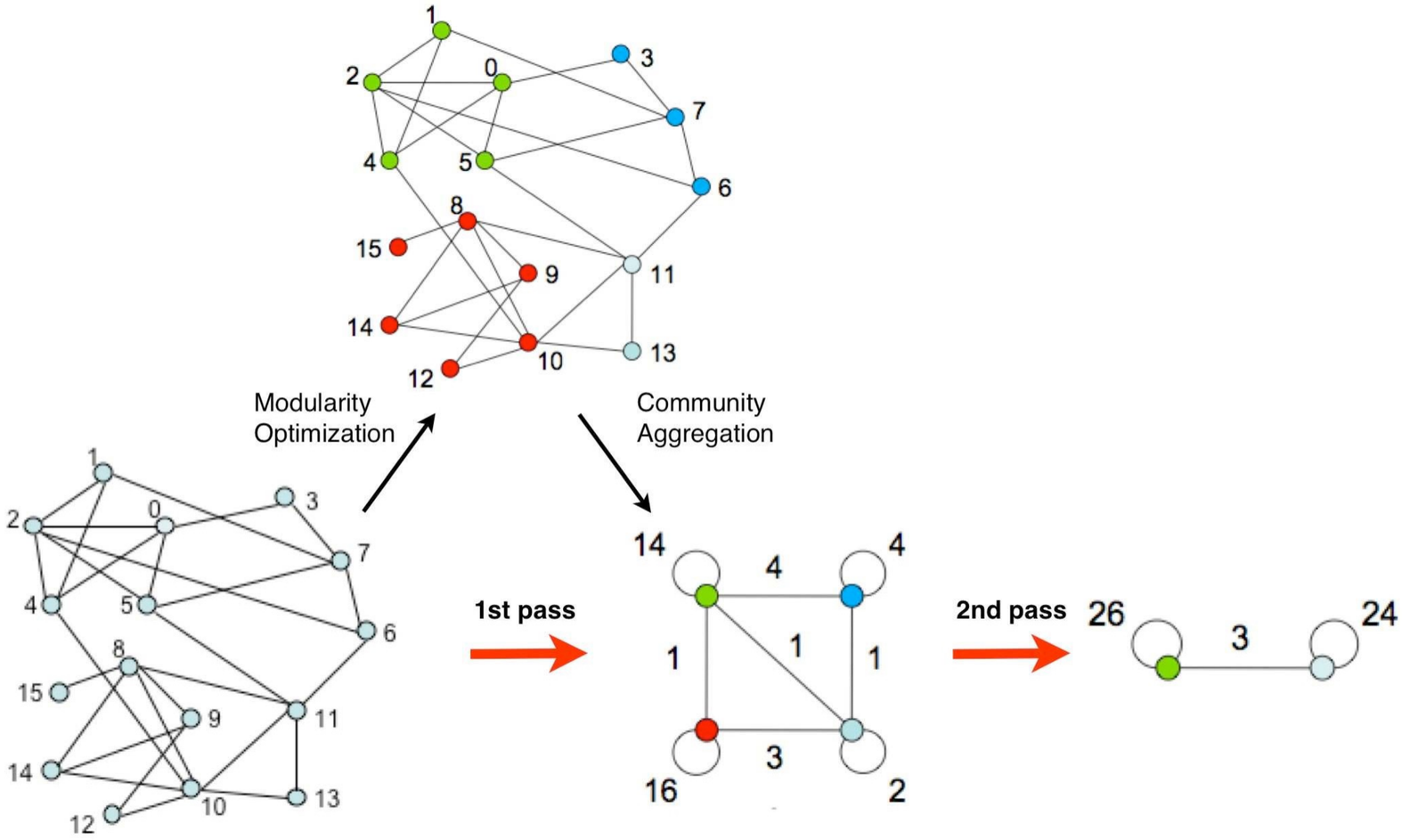
The communities obtained in the first phase are contracted into super-nodes, and the

network is created accordingly:

- Super-nodes are connected if there is at least one edge between the nodes of the corresponding communities
- The weight of the edge between the two super-nodes is the sum of the weights from all edges between their corresponding communities

 Phase 1 is then run on the super-node network

Louvain algorithm



Louvain algorithm

Limits:

- Like Newman's algorithm, it is a greedy method, in that it tries to find the best modularity partition at each level of agglomeration. Therefore it yields solutions with sub-optimal modularity
- The final partition depends on the order in which nodes are visited

Big plus:

- The algorithm is very fast because, after the first iteration, the successive transformations shrink the network very quickly and typically only a handful of partitions are generated. It can detect communities in networks with millions of nodes and links

Louvain algorithm

There currently is no function in NetworkX for the Louvain algorithm. However, it is possible to implement the algorithm using the `community` module

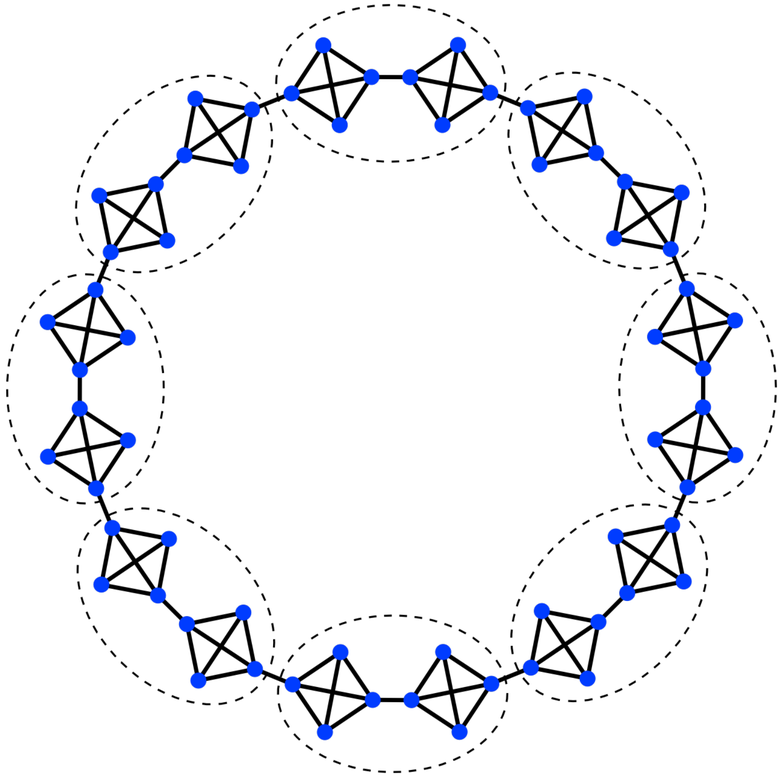
```
# download community module at
# github.com/taynaud/python-louvain
import community

# returns the partition with largest modularity
partition_dict = community.best_partition(G)
```

Modularity optimization: limits

- The maximum modularity tends to be larger on larger networks, so the measure cannot be used to compare the quality of partitions across networks of different sizes
- The maximum modularity of random networks without group structure (e.g., Erdős–Rényi) can attain fairly large values
- The maximum modularity does not necessarily correspond to the best partition. Communities smaller than a certain size may not be detected (**resolution limit**)

Modularity: resolution limit



The natural partition is the one where the communities are the cliques but modularity is larger for the partition where pairs of cliques are merged

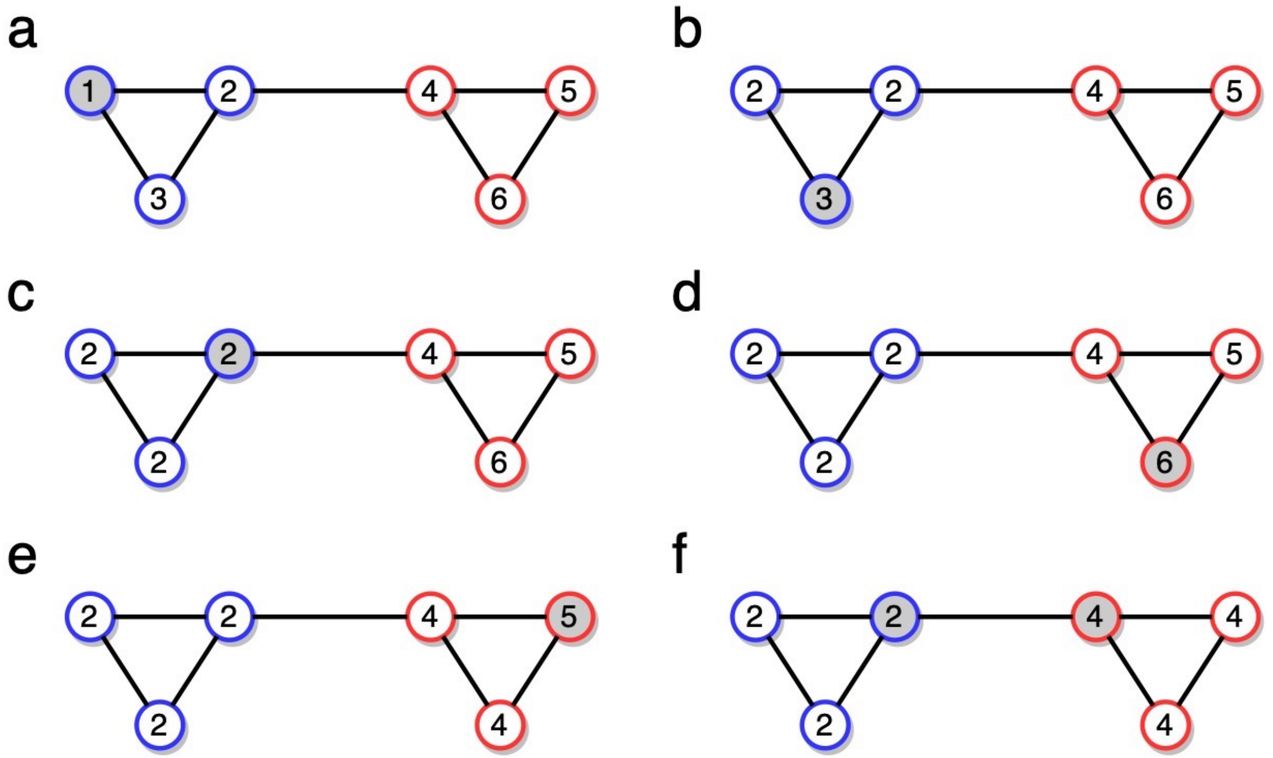
- Possible solution: tuning the resolution of the method by inserting a parameter in the modularity formula (**multiresolution modularity optimization**)
- Two problems:
  - Computationally intensive: modularity has to be optimized for multiple choices of the resolution parameter
  - A criterion is needed to decide which value of the resolution parameter is most suitable for a given network

Label propagation

- **Principle:** neighbors of a node usually belong to the same community
- **Procedure:**
  1. Start: each node is assigned to a different community. Each node is given a different label
  2. A sweep is performed over all nodes, in random order: each node takes the label shared by the majority of its neighbors. If there is no unique majority, one of the majority labels is picked at random
  3. If every node has the majority label of its neighbors (**stationary state**), stop. Else repeat step 2
- Communities are defined as groups of nodes having identical labels in the stationary state

Label propagation

- Labels propagate during the process: most labels disappear, others dominate
- The algorithm typically converges after a small number of iterations, fairly independent of network size
- In the final partition, each node has more neighbors in its own community than in any other, so **each cluster is a strong community** (less stringent definition)



Label propagation: limits

- **Order-dependent**: The algorithm does not deliver a unique solution; the outcome depends on the order in which the nodes are visited in each sweep

- **Ties-dependent:** Different partitions are also the result of the many ties encountered along the process, which can be broken in different ways depending on the sequence of random numbers

Label propagation

- The algorithm does not need any information about the number and the size of the communities
- It is parameter-free
- It is simple to implement and very fast: networks with millions of nodes and links can be partitioned this way
- If community labels are known for some of the nodes, they can be used as seeds in the initial partition

```
# returns the partition found by the label propagation method
partition = nx.community.asyn_lpa_communities(G)
```

How to generate a graph having community structure?

© 2020 F. Menczer, S. Fortunato & C.A. Davis. cambridgeuniversitypress.github.io/FirstCourseNetworkScience

Exercise

Invent a variant of the ER model that generates graphs having **two communities**, but not necessarily disconnected

Standard ER Model:

- Generate N nodes
- For each of the $N(N-1)/2$ pairs of nodes:
- Perform a Bernoulli trial with probability p
- If the trial succeeds, connect those nodes

Pin board: https://upfbarcelona.padlet.org/chato/tt20-community-structure-yhk7nozup8xbiejb

© 2020 F. Menczer, S. Fortunato & C.A. Davis. cambridgeuniversitypress.github.io/FirstCourseNetworkScience

Stochastic block modeling

- **Basic assumption:** the network has been generated by some model.
- Models are characterized by one or more parameters
- **Question:** for which values of the parameters does the model produce networks that most closely resemble the one we are investigating?
- **Example:** if we know/assume that the network has been generated by an Erdős–Rényi random graph, for which value of the link probability do we get graphs most similar to the network?

Stochastic block model

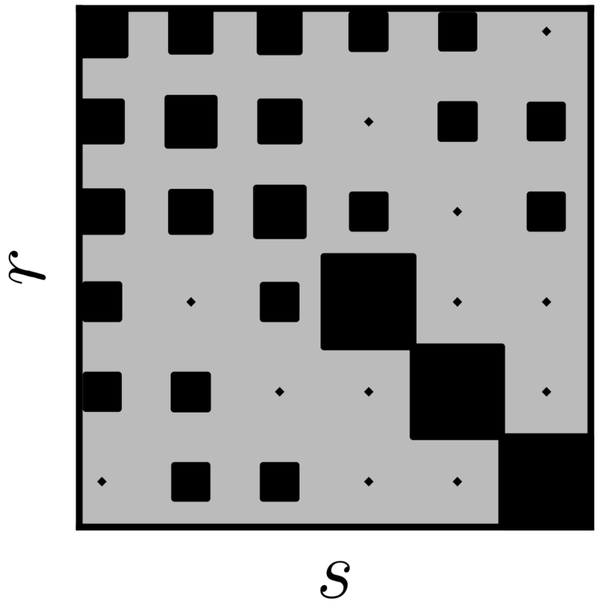
- **Our focus:** community structure!
- We need to look for models that generate networks with built-in communities
- **Stochastic block models** (**SBM**) are the most important class of models generating networks with communities
- **Principle:** nodes are divided into groups and the probability that two nodes are connected is determined by the groups to which they belong

Stochastic block model

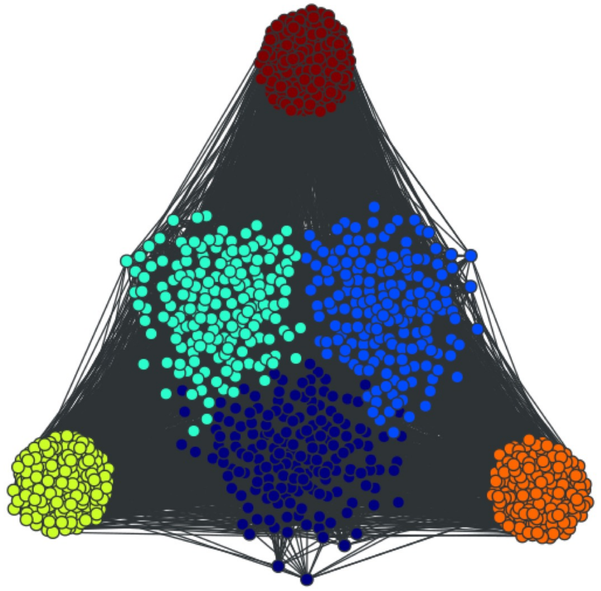
- q groups, g_i label of group of node i
- The probability to form a link between nodes i and j is

$$P(i \leftrightarrow j) = p_{g_i g_j}$$

- The **stochastic block matrix** is the $q \times q$ matrix whose element kl is the link probability p_{kl}
- The diagonal element p_{kk} is the link probability between pairs of nodes in group k

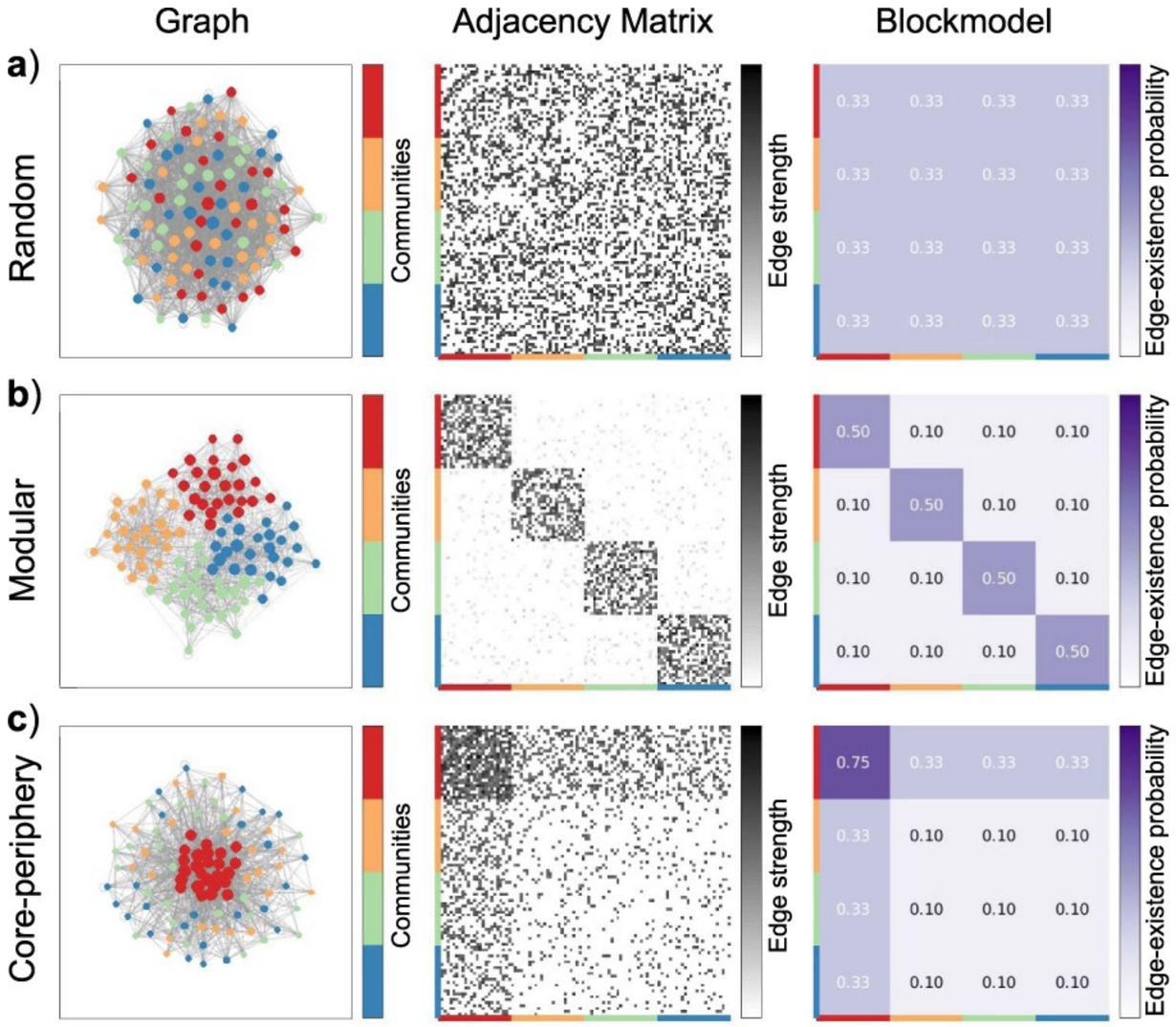


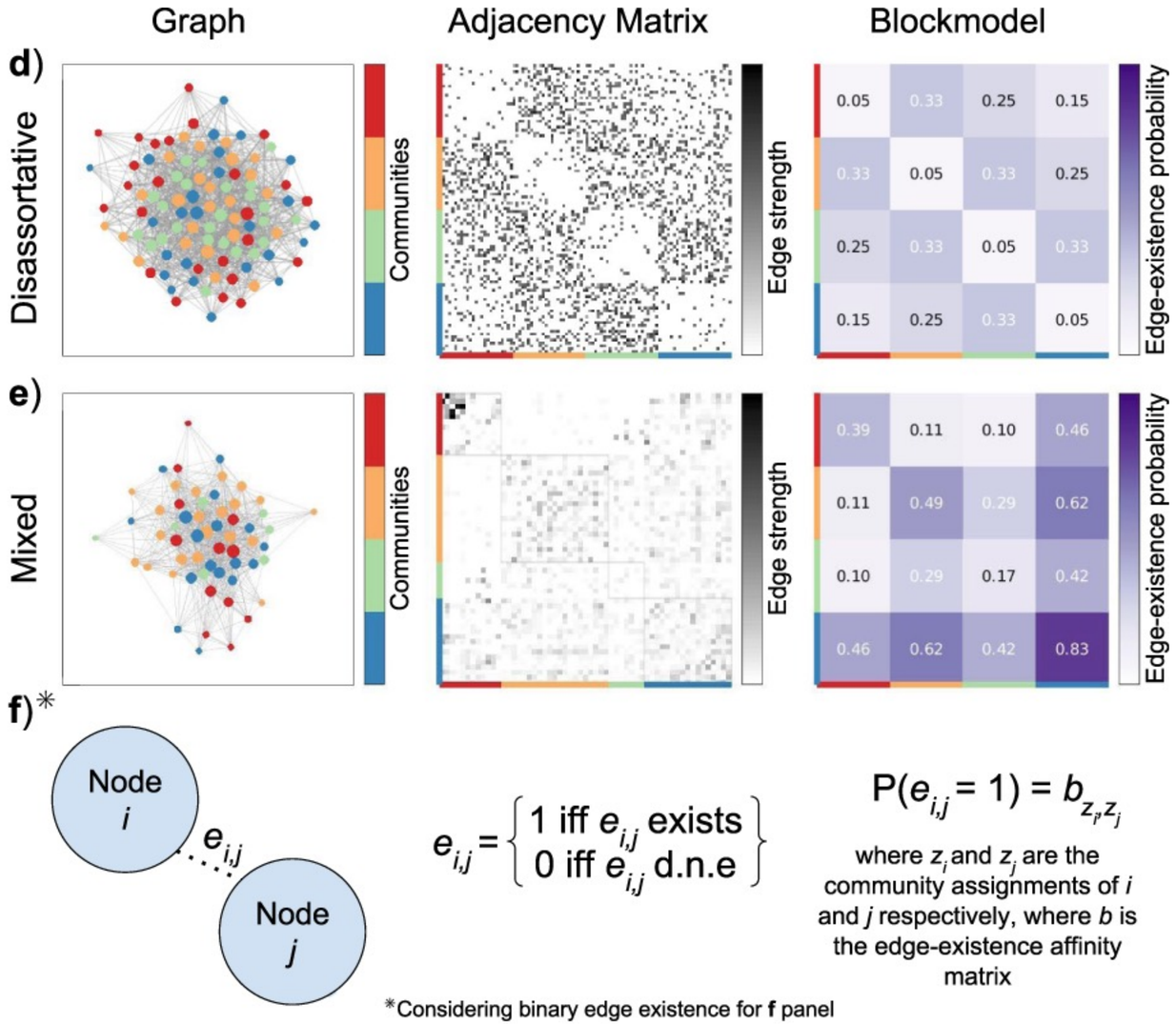
Stochastic block matrix



Network generated by SBM

Stochastic block model





$$e_{i,j} = \begin{cases} 1 \text{ iff } e_{i,j} \text{ exists} \\ 0 \text{ iff } e_{i,j} \text{ d.n.e} \end{cases}$$

$$P(e_{i,j} = 1) = b_{z_i z_j}$$

where z_i and z_j are the community assignments of i and j respectively, where b is the edge-existence affinity matrix

*Considering binary edge existence for **f** panel

Stochastic block model

- If $\forall r, s = 1, \ldots, q$ with $r \neq s$ we have $p_{rr} > p_{rs}$, we have **community structure:** links are more likely within than between blocks
- If $\forall r, s = 1, \ldots, q$ with $r \neq s$ we have $p_{rr} < p_{rs}$, we have **multipartite structure:** links are more likely between than within blocks
- If $q = 2$ and $p_{11} \gg p_{12} \gg p_{22}$, we have **core-periphery structure**: the nodes in the first block (core) are relatively well-connected amongst themselves as well as to a peripheral set of nodes that interact very little among themselves
- If $p_{rs} = p \ \forall r, s = 1, \ldots, q$ we recover the **classic random network**: any two nodes have identical probability to be connected; there is no group structure

Fitting a stochastic block model to a network

- **Question:** how shall we fit the model to the network being considered?
- **Answer:** maximize the likelihood that, for a given partition of the network, a SBM reproduces the placement of links between the nodes
- The best partition is the one corresponding to the largest value of the likelihood
- **Problem:** the standard SBM does not describe well real networks, because it ignores degree heterogeneity: networks generated with the model usually have nodes with similar degrees
- **Solution:** the **degree-corrected stochastic block model** (**DCSBM**) uses the actual degrees of the nodes of the network

Stochastic block modeling: limits

- It is necessary to provide as input the number of clusters, which is usually unknown. In fact, a straight maximization of the likelihood over the whole set of possible partitions yields a trivial division into *N* groups of one node each. However, there are techniques to estimate the number of clusters

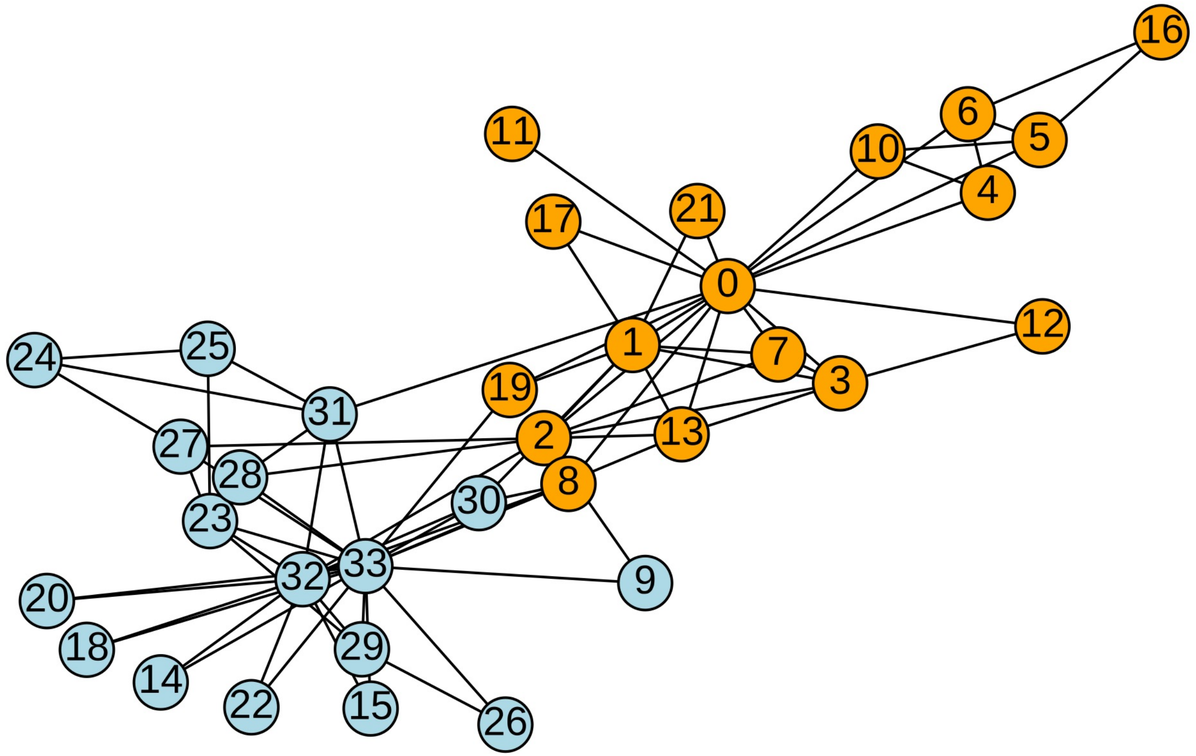
- The greedy algorithm for the maximization of the likelihood gets stuck at suboptimal solutions. To improve the result, it helps to run the algorithm several times with different random initial conditions and select the partition with highest likelihood across all runs

Method evaluation

- **Problem:** How can we tell how good a clustering algorithm is?
- **Solution:** The natural way to evaluate a method is to check whether it is able to find clusters in **benchmark graphs**, i.e., networks known to have a “natural” community structure
- Two classes of benchmarks:
  - **Artificial benchmarks:** computer-generated networks, built via some model
  - **Real benchmarks:** real networks in which the communities are suggested by the history of the system or by attributes of the nodes

Real benchmarks

- **Classic example:** Zachary's karate club network
- Network of social relationships between members of a karate club in the US
- Following a disagreement between the instructor (node 0) and the president (node 33), the club split in two different groups
- The task is to identify those groups (identified by the node colors) using a clustering algorithm




```
G = nx.karate_club_graph()
```

Summary

Things to remember

- Girvan-Newman algorithm
- Modularity definition: null model, resolution limits
- Louvain algorithm
- Label propagation algorithm
- Stochastic block model

Sources

- F. Menczer, S. Fortunato, C. A. Davis (2020). A First Course in Network Science – Chapter 06

- URLs cited in the footer of slides

© 2020 F. Menczer, S. Fortunato & C.A. Davis. cambridgeuniversitypress.github.io/FirstCourseNetworkScience