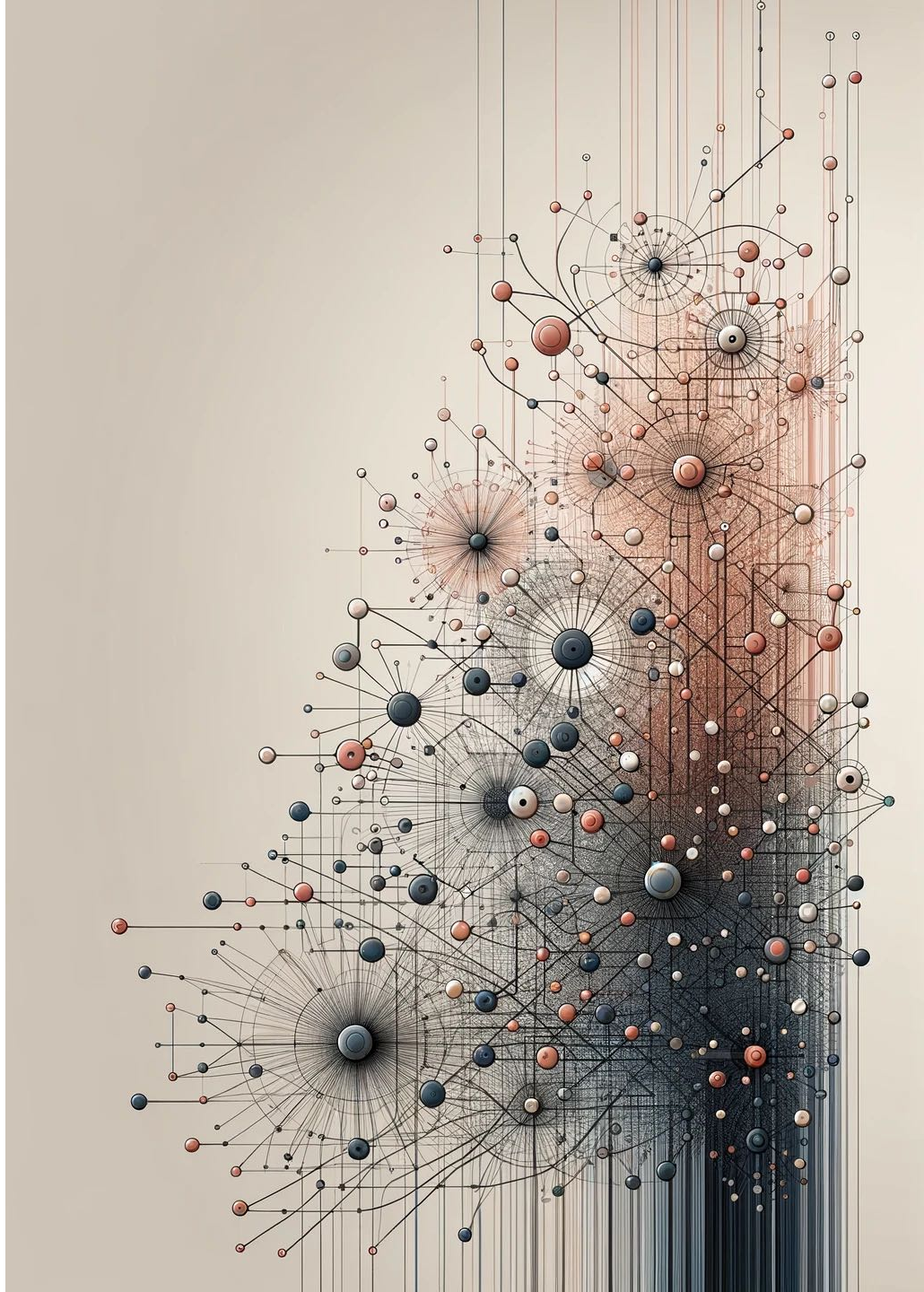
UNIVERSITÀ DI TORINO

Analisi e Visualizzazione delle Reti Complesse

NS13 - Community Detection

Prof. Rossano Schifanella



UNIVERSITÀ DI TORINO

Outline

- Community detection
  - Girvan-Newman
  - Modularity and modularity optimization
  - Newman's Greedy Algorithm
  - Louvain Algorithm
  - Modularity's limits
  - Label propagation
  - Stochastic block modeling

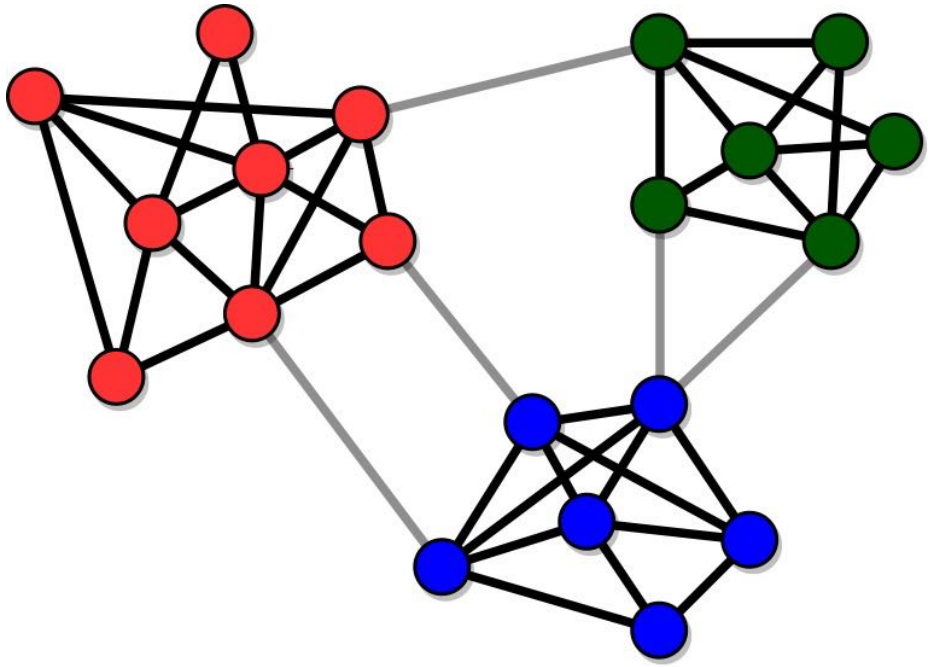
UNIVERSITÀ DI TORINO

Community detection

UNIVERSITÀ DI TORINO

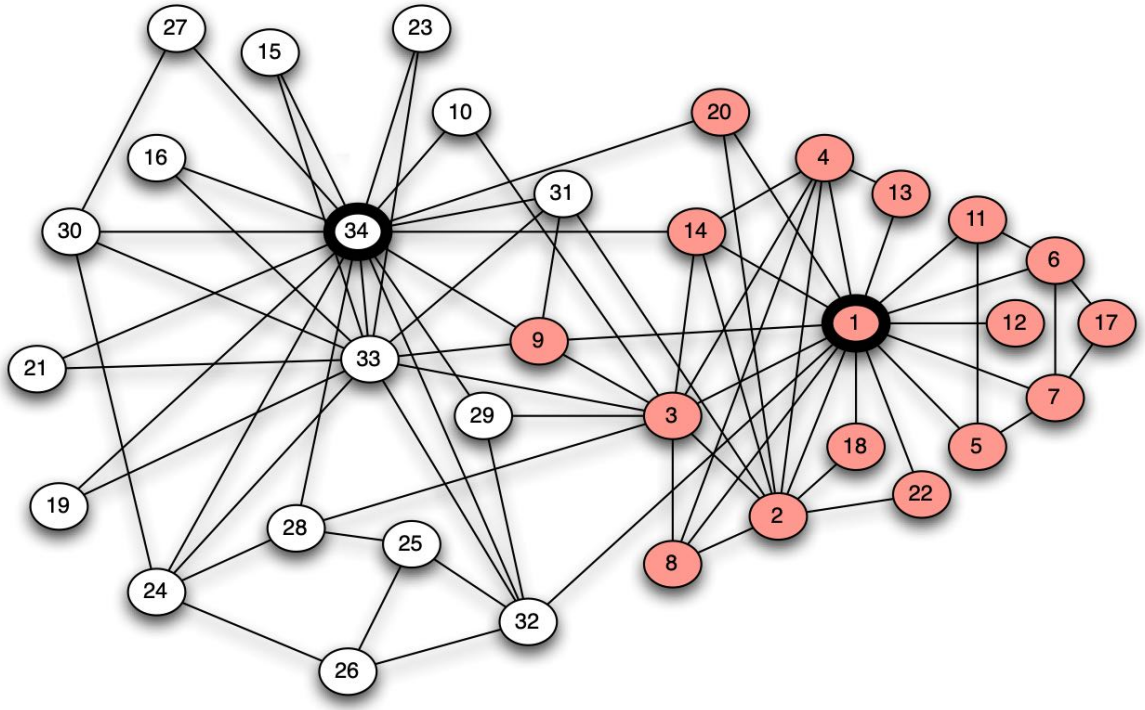
Community detection

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



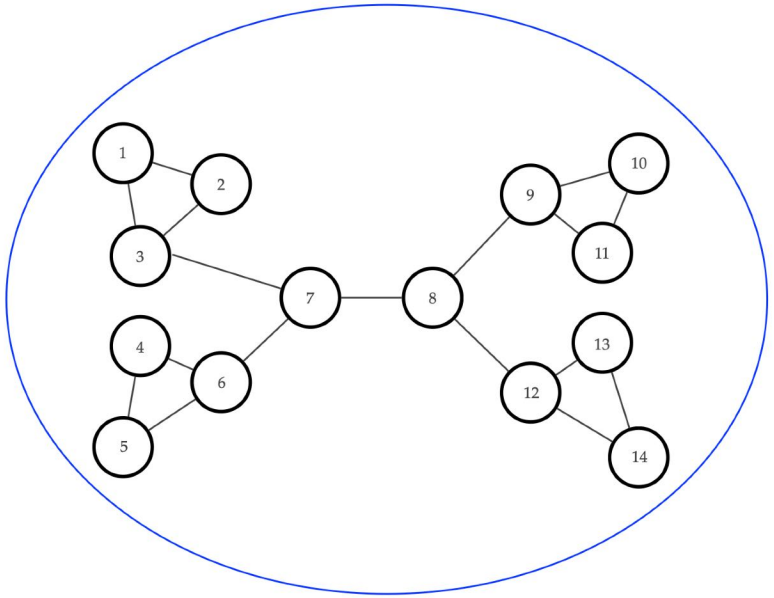
Example: Zachary Club

- Can I use the network structure to predict the fault line?



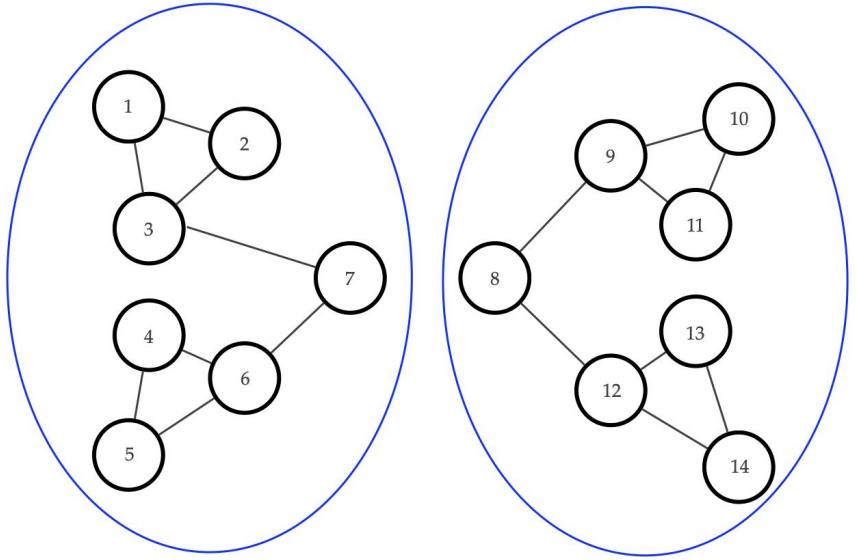
General approaches

- **Divisive**
  - **top-down**
  - goal: remove "spanning links"



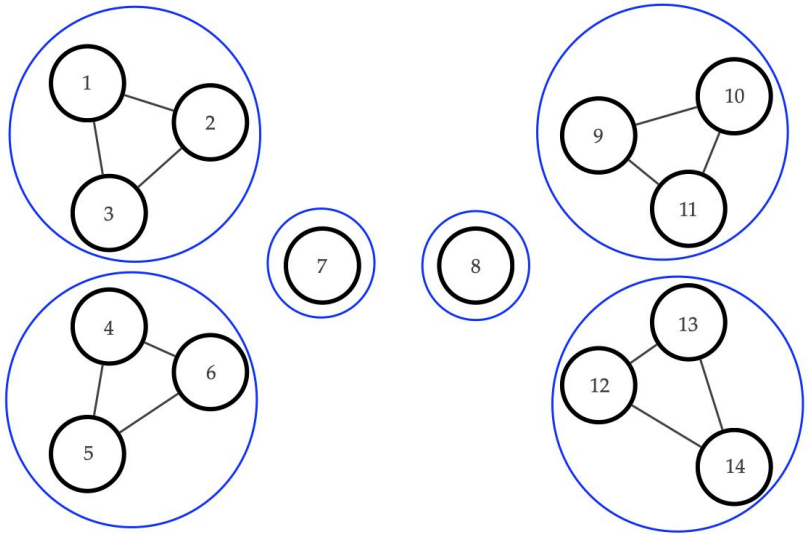
General approaches

- **Divisive**
  - **top-down**
  - goal: remove "spanning links"



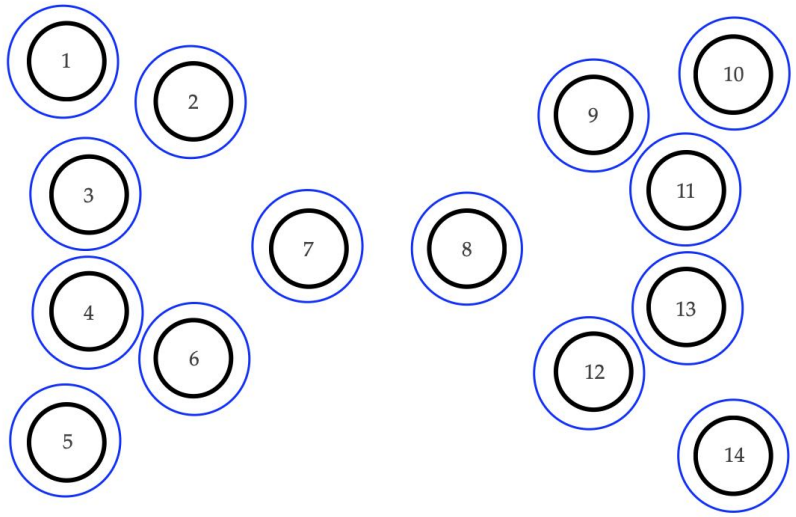
General approaches

- **Divisive**
  - **top-down**
  - goal: remove "spanning links"



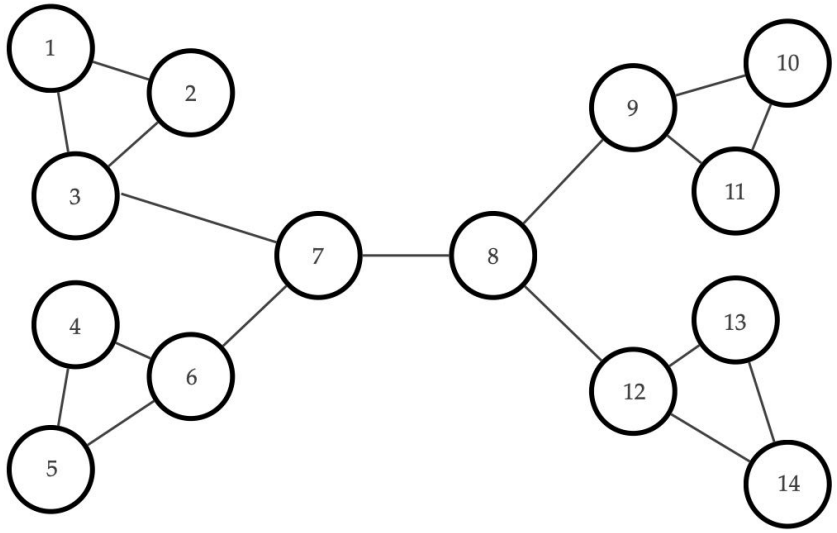
General approaches

- **Divisive**
  - **top-down**
  - goal: remove "spanning links"



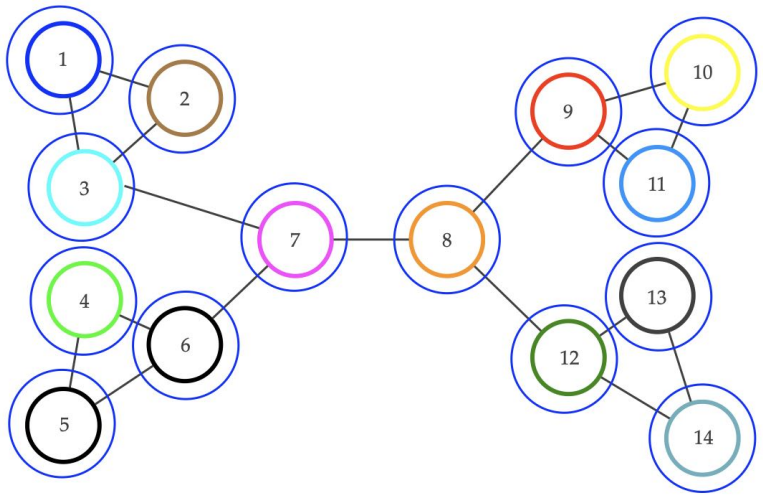
General approaches

- **Agglomerative**
  - **bottom-up**
  - goal: form groups merging nodes with other nodes



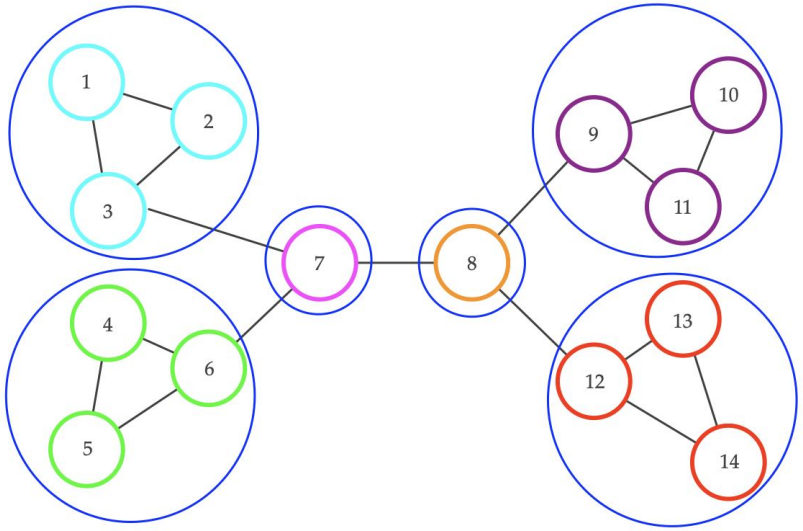
General approaches

- **Agglomerative**
  - **bottom-up**
  - goal: form groups merging nodes with other nodes



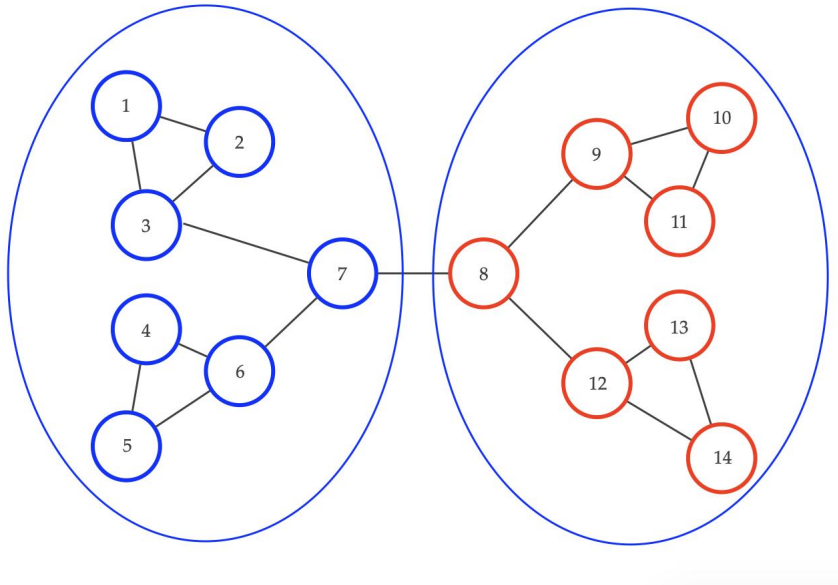
General approaches

- **Agglomerative**
  - **bottom-up**
  - goal: form groups merging nodes with other nodes



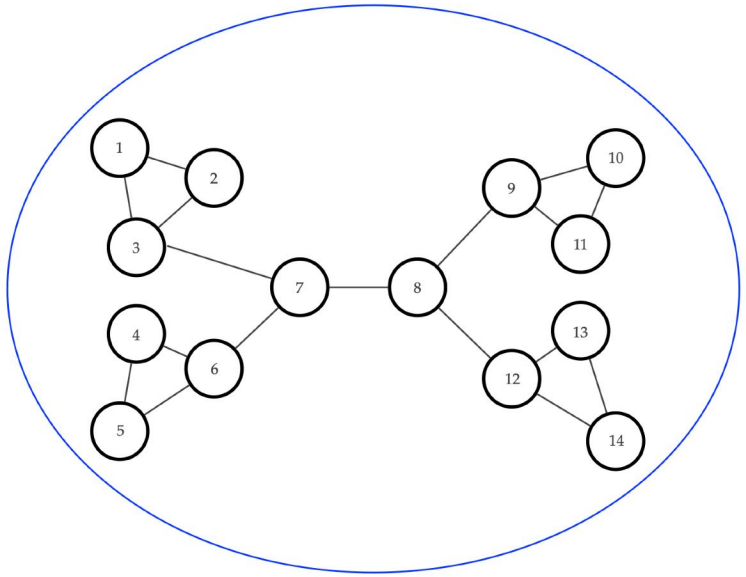
General approaches

- **Agglomerative**
  - **bottom-up**
  - goal: form groups merging nodes with other nodes



General approaches

- **Agglomerative**
  - **bottom-up**
  - goal: form groups merging nodes with other nodes



Which edge to remove first?

- In divisive methods, we need a rule to find **spanning edges**
- let's exploit lessons learnt from **robustness**
- (local) bridges are good candidates: if removed, distances will be longer $\rightarrow \infty$

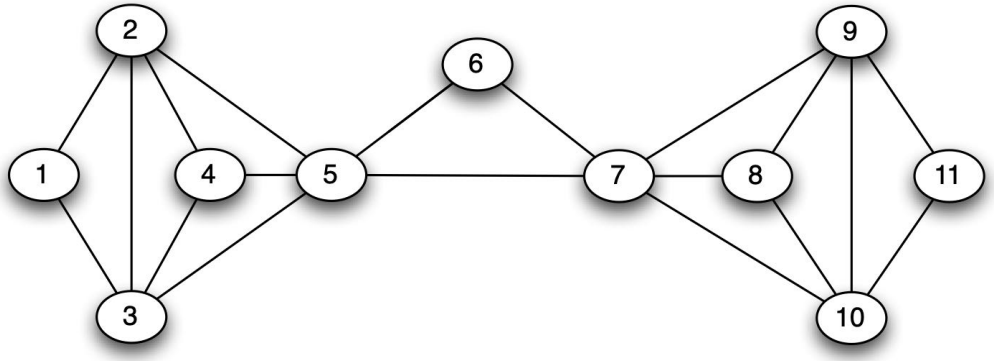
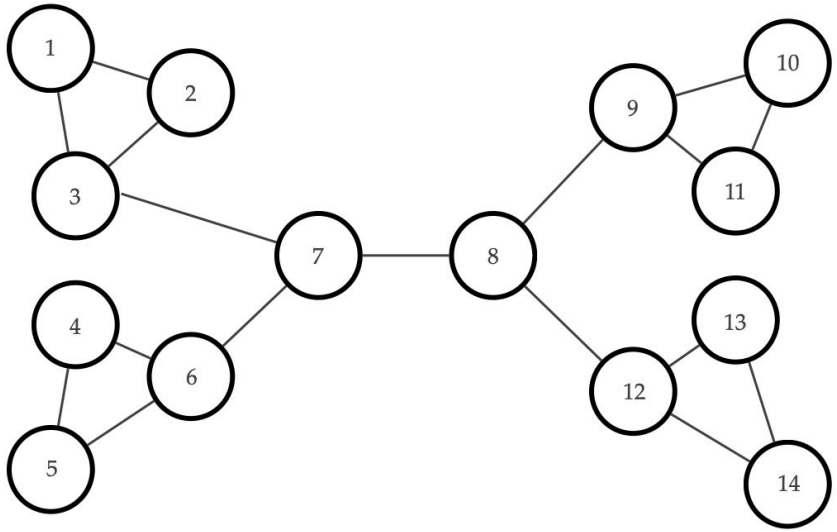


Figure 3.15: A network can display tightly-knit regions even when there are no bridges or local bridges along which to separate it.

Betweenness of an edge

- how many shortest paths will be affected if edge (7,8) is removed?
- betweenness is probably a better candidate for removal
- b_{ij} = betweenness of an edge $(i, j) =$ number of shortest paths that cross through (i, j)



$$b_{7,8} = 49 \quad b_{3,7} = 33$$

Observations on edge's betweenness

- **high betweenness:** correlated to local bridges (and, according to Granovetter, to weak ties)
- inversely correlated to neighborhood overlap and to the clustering coefficient of its endpoints
- what about structural holes?
- linked to some notion of **traffic** or **flow**

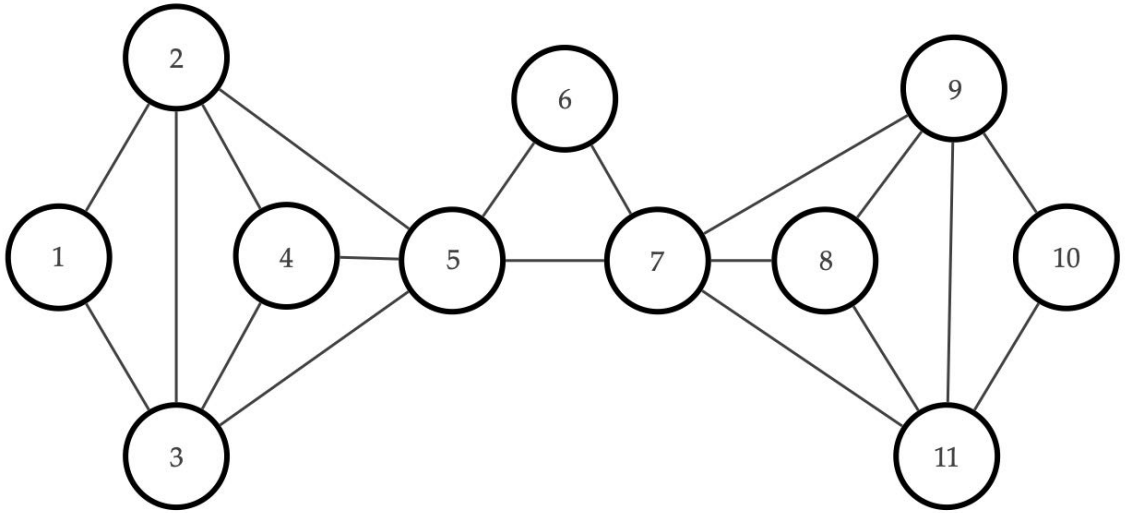
The Girvan-Newman method

Procedure:

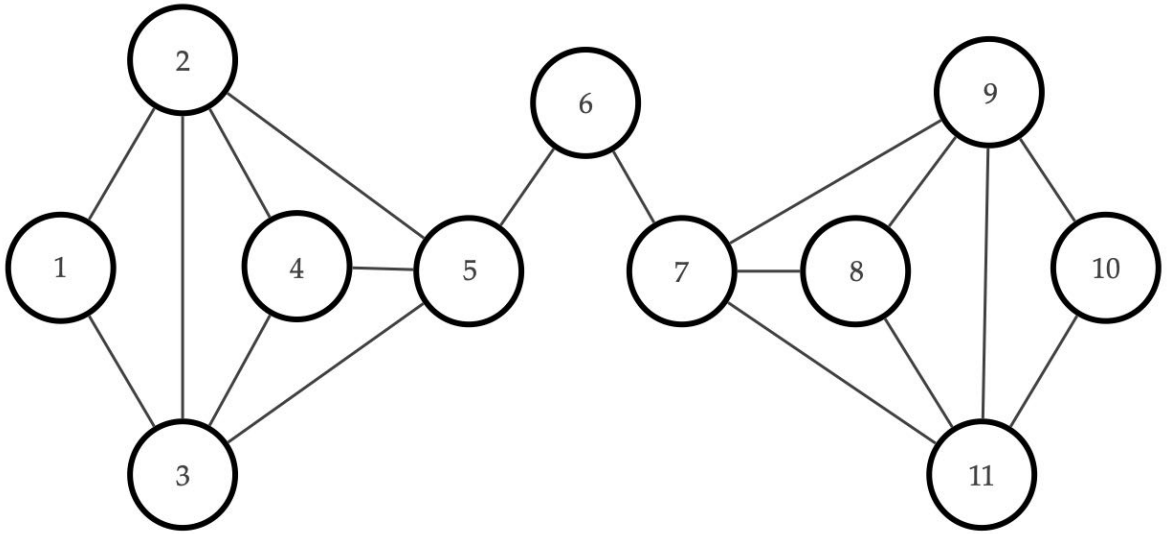
1. Calculate betweenness for all the edges
2. Find edge(s) with highest betweenness
3. Remove those edges
4. **if** satisfied **or** no more edges **return** components as clusters
5. **else** goto 1.

- Reading material:
  - Newman, Mark EJ, and Michelle Girvan. "Finding and evaluating community structure in networks." Physical review E 69.2 (2004): 026113

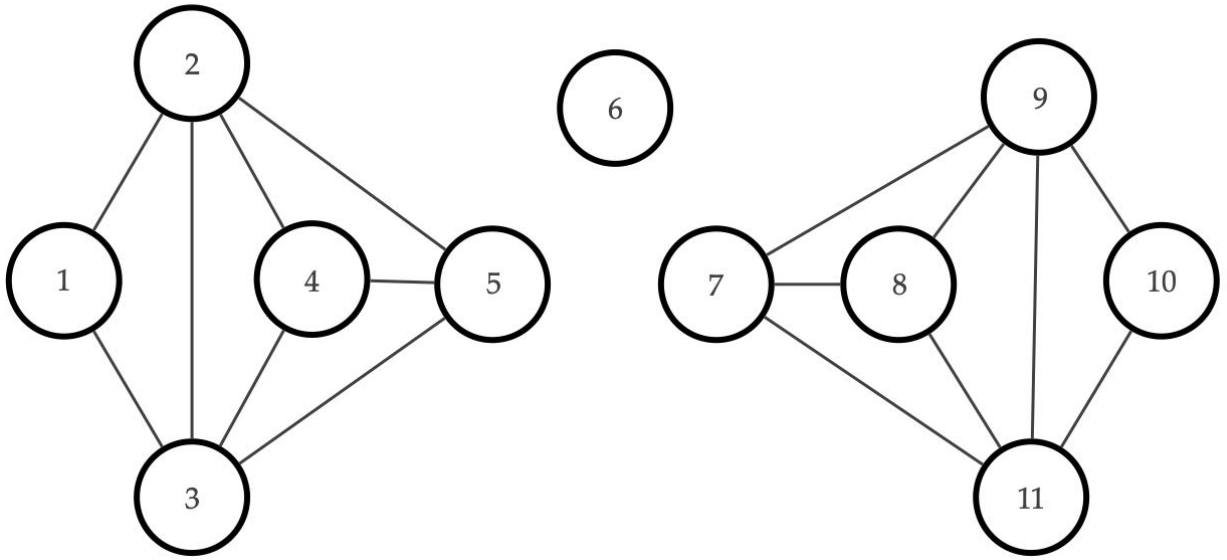
The Girvan-Newman method



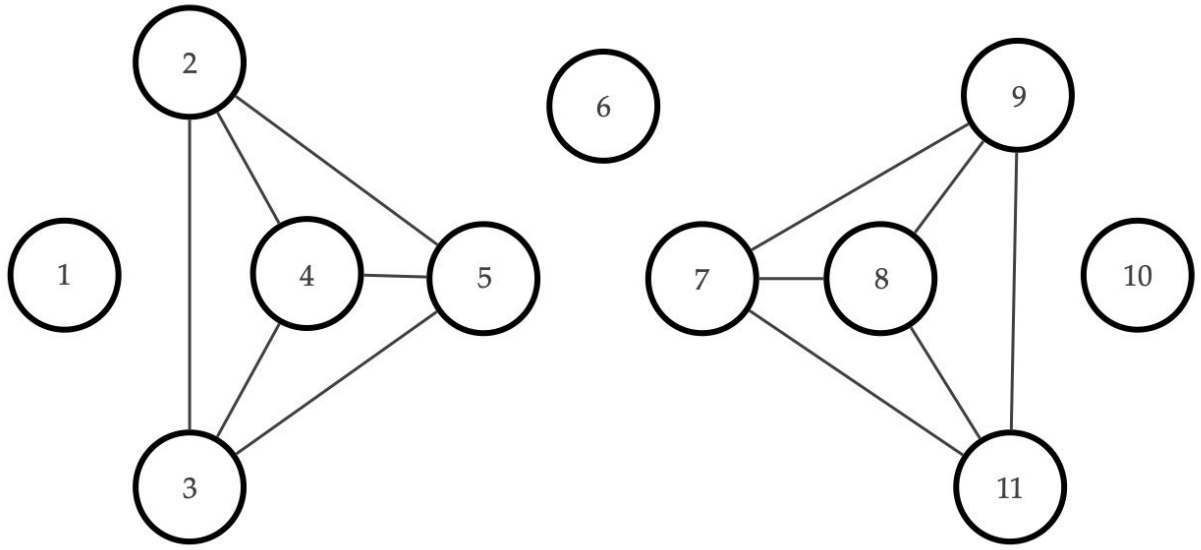
The Girvan-Newman method



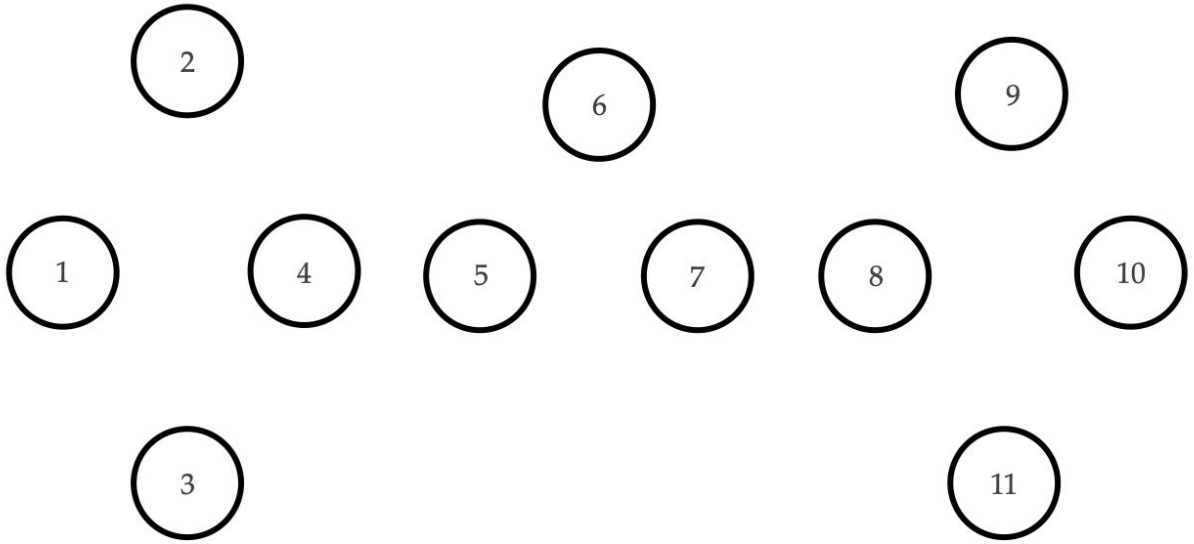
The Girvan-Newman method



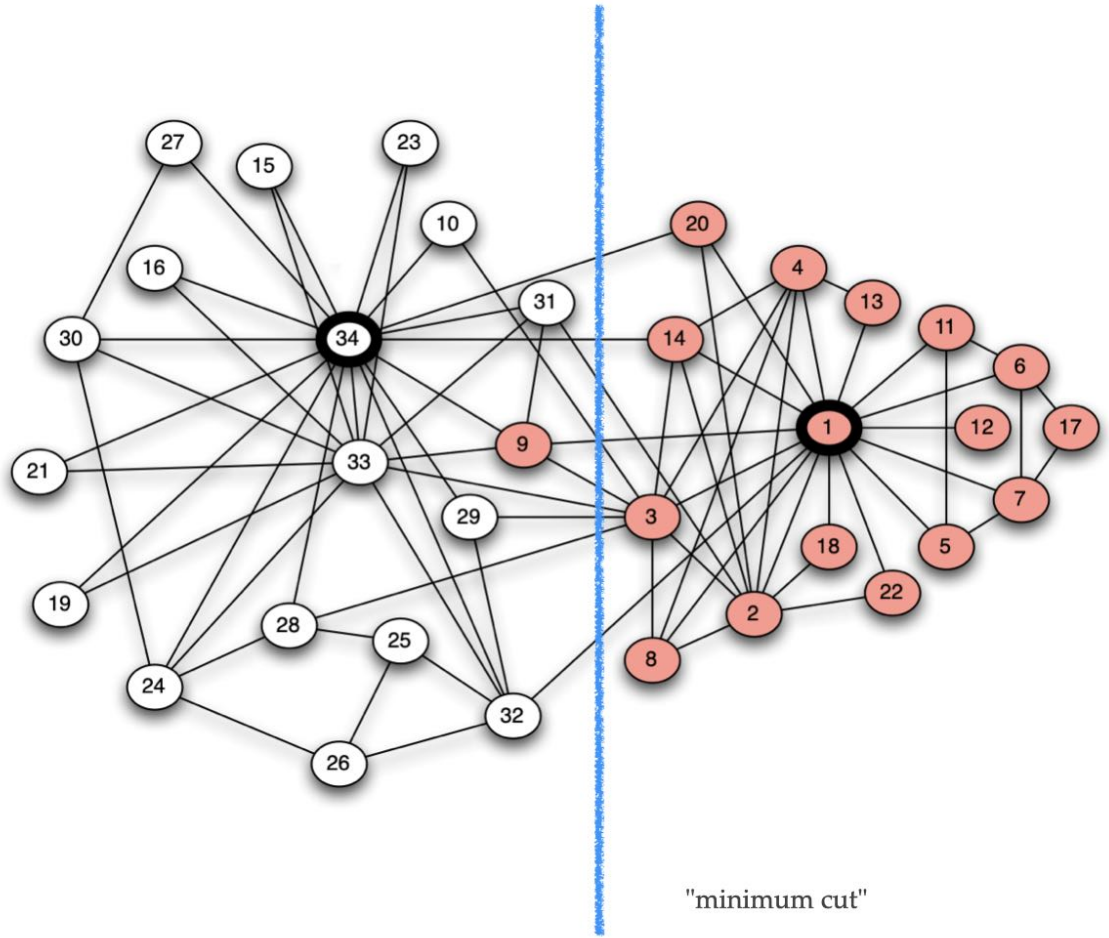
The Girvan-Newman method



The Girvan-Newman method



Applying Girvan-Newman to Zachary Club



Observations and questions on Girvan-Newman

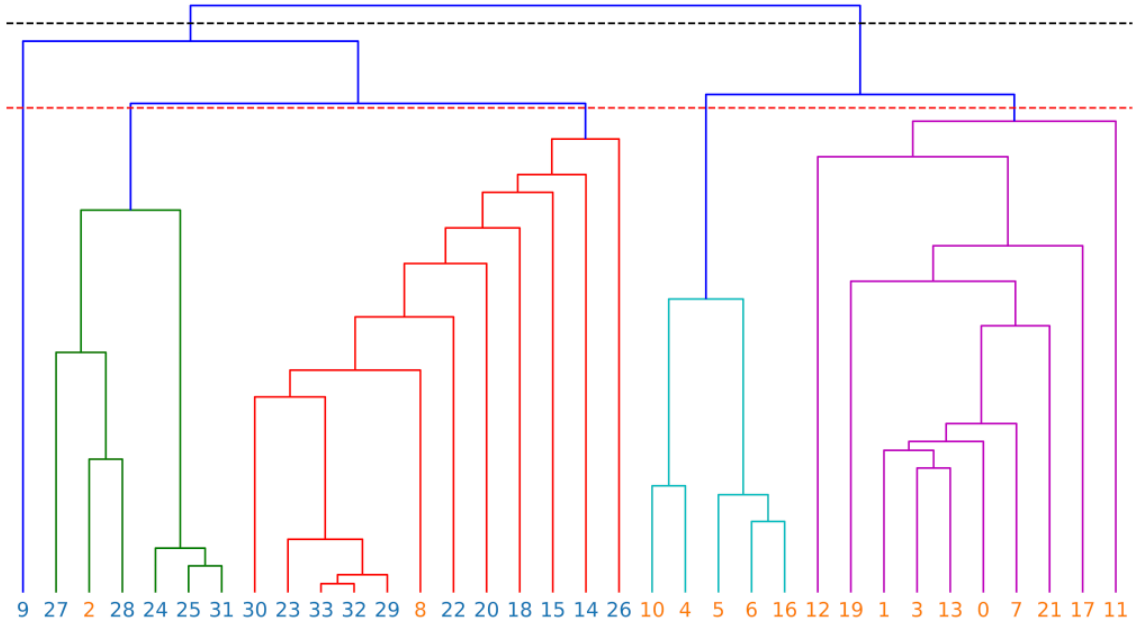
- We do not need to find **local** bridges
- betweenness is an excellent approximation that is calculated through a **global** property of the network
- How to efficiently calculate edge's betweenness?
  - see the previous class!
- When do I stop?
  - many criteria, more later

Girvan-Newman algorithm

- The recalculation of the betweenness at each iteration is necessary, but it makes the algorithm slow
- On networks with strong community structures, 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**



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 a 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

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 are meaningful, and how can they be selected?

In NetworkX:

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

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

Homophily test and modularity

- Recall **homophily test**: if p is the fraction of nodes in group A and q the fraction of nodes in group B, then we have a signal of homophily if the **number of actual cross groups edges** $< 2pq$
- Modularity follows the same principle, but:
  - we can have more than two groups
  - we need to make our comparison with a **degree preserving randomization**, like the one we get from the Configuration Model

Modularity: community-based representation

We can rewrite modularity in terms of communities:

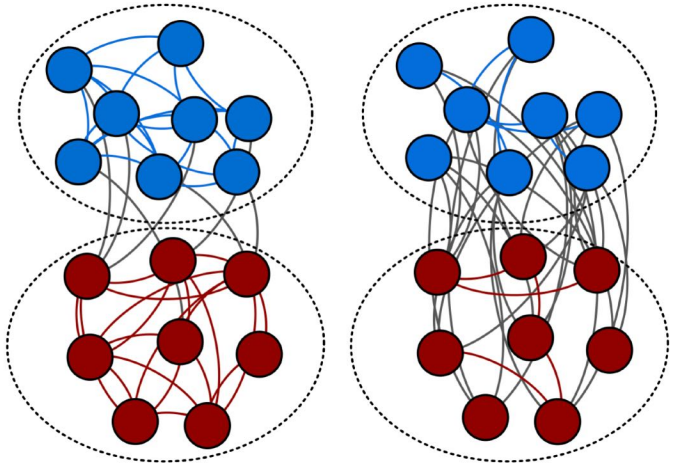
$$Q = \frac{1}{L} \sum_{C} \left(L_C - \frac{k_C^2}{4L} \right)$$

Where:

- L is the number of links in the network
- L_C is the number of internal links in community C
- k_C is the sum of degrees of all nodes in community C
- $\frac{k_C^2}{4L}$ is the expected number of internal links in community C in our random model

Modularity: understanding the null model

- The term $\frac{k_C^2}{4L}$ represents our expectation in a **configuration model** (random network with same degree sequence)
- Step-by-step derivation:
  - Probability of selecting first stub from community C: $p_1 = \frac{k_C}{2L}$
  - Probability of selecting second stub from community C: $p_2 = \frac{k_C}{2L}$
  - Probability of forming an internal edge in C: $p_1 \times p_2 = \frac{k_C^2}{4L^2}$
  - Expected number of internal edges in C: $p_1 \times p_2 \times L = \frac{k_C^2}{4L}$
- A **positive/negative** modularity indicates **more/fewer** internal edges than expected by random chance



Modularity: features

- **Q = 0**: The community structure is no better than random
- **Q < 0**: The community structure is worse than random (communities are less densely connected internally than would be expected by chance)
- **Q > 0**: The network has more connections within communities than would be expected in a random network
- **Q ≈ 0.3-0.7**: Typical range for networks with strong community structure
- **Q < 1**: Theoretical upper bound (practically never reached)

In NetworkX:

```
modularity = nx.community.quality.modularity(G,partition)
```

Modularity: An alternative definition

- For a partition of a network into communities, modularity measures the **density of links inside communities compared to links between communities**
- Mathematically, it's defined as:

$$Q = \frac{1}{2m} \sum_{i,j} \left[A_{ij} - \frac{k_i k_j}{2m} \right] \delta(c_i, c_j)$$

Where:

- A_{ij} is the adjacency matrix element (1 if nodes i, j are connected, 0 otherwise)
- k_i, k_j are the degrees of nodes i and j
- $2m$ is the total number of edges in the network
- $\delta(c_i, c_j)$ is 1 if nodes i and j belong to the same community, 0 otherwise
- $\frac{k_i k_j}{2m}$ is the expected number of edges between nodes i and j in a random network

Equivalence of Modularity Definitions

Consider the two mathematically equivalent formulations of modularity:

$$\text{Formal definition: } Q = \frac{1}{2m} \sum_{i,j} \left[A_{ij} - \frac{k_i k_j}{2m} \right] \delta(c_i, c_j)$$

$$\text{Community-based definition: } Q = \sum_{C} \left[\frac{L_C}{L} - \left(\frac{k_C}{2L} \right)^2 \right]$$

Where $L = m$ is the total number of links in the network.

Demonstration of Equivalence

Let's prove these formulations are equivalent:

1. Group the summation in the formal definition by community:
   - For nodes i, j in the same community C, $\delta(c_i, c_j) = 1$
   - The sum $\sum_{i,j \in C} A_{ij} = 2L_C$ (each internal edge is counted twice)
   - The sum $\sum_{i \in C} k_i = k_C$ (total degree of community C)
2. Rewriting the formal definition:

$$Q = \frac{1}{2m} \sum_C \left[\sum_{i,j \in C} A_{ij} - \sum_{i,j \in C} \frac{k_i k_j}{2m} \right]$$

$$Q = \frac{1}{2m} \sum_C \left[2L_C - \frac{k_C^2}{2m} \right]$$

$$Q = \sum_C \left[\frac{L_C}{m} - \frac{k_C^2}{4m^2} \right]$$

1. Substituting $m = L$:

$$Q = \sum_C \left[\frac{L_C}{L} - \left(\frac{k_C}{2L} \right)^2 \right] = \frac{1}{L} \sum_C \left(L_C - \frac{k_C^2}{4L} \right)$$

Thus, the formal node-based definition and the community-based definition are mathematically equivalent.

Modularity: extensions

Directed networks

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

Where:

- L is the number of links in the network
- L_C is the number of internal links in community C
- k_C^{in} is the total in-degree of nodes in community C
- k_C^{out} is the total out-degree of nodes in community C

Modularity for directed networks: explanation

- In directed networks, links have direction, so we must account for **in-degree** and **out-degree**
- The null model changes:
  - The probability of a directed link from node i to j becomes $\frac{k_i^{out} k_j^{in}}{L}$
- For a community C:
  - The probability of selecting an outgoing stub from C is $\frac{k_C^{out}}{L}$
  - The probability of selecting an incoming stub to C is $\frac{k_C^{in}}{L}$
  - Expected internal links: $\frac{k_C^{in} k_C^{out}}{L}$
- The modularity compares actual internal links (L_C) vs. expected internal links

Modularity: extensions

Directed and weighted networks

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

Where:

- W is the total weight of network links
- W_C is the total weight of internal links in the community C
- s_C^{in} is the total in-strength of nodes in community C
- s_C^{out} is the total out-strength of nodes in community C

Weighted directed modularity: explanation

- For weighted directed networks, we need to consider:
  - Edge weights (strength instead of degree)
  - Edge directions (in/out connections)
- The weighted directed null model:
  - Probability of an edge from i to j proportional to $\frac{s_i^{out} s_j^{in}}{W}$
  - W is total weight sum in the network
- For a community C:
  - s_C^{in} = sum of in-strengths of all nodes in C
  - s_C^{out} = sum of out-strengths of all nodes in C
  - Expected internal weight: $\frac{s_C^{in} s_C^{out}}{W}$

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!** The number of possible ways to partition a network grows faster than exponentially with the number of nodes
- Finding the partition with maximum modularity is NP-hard

Clauset-Newman-Moore's greedy algorithm

Core idea: Hierarchical agglomeration with greedy modularity optimization

Intuition:

- For communities i and j, merging them changes modularity by:

$$\Delta Q = \frac{e_{ij}}{L} - \frac{k_i k_j}{2L^2}$$

Procedure:

1. **Initialize**: Each node starts as its own community (N communities)
2. **Calculate**: For each pair of connected communities, compute modularity gain ΔQ if merged
3. **Merge**: Combine the pair with largest ΔQ
4. **Update**: Recalculate ΔQ values for the new community and its neighbors
5. **Repeat**: Steps 3-4 until all nodes are in one community
6. **Select**: Choose the partition with maximum modularity from all iterations

- Reading material:
  - Clauset, A., Newman, M. E., & Moore, C. "Finding community structure in very large networks." Physical Review E 70(6), 2004.

In NetworkX:

```
partition = nx.community.greedy_modularity_communities(G)
```

Clauset-Newman-Moore's greedy algorithm: limits

- **Local optima problem**: As a greedy method, it makes locally optimal choices at each step but can miss the global optimum
- **Resolution limit**: Cannot detect communities smaller than a scale dependent on network size
  - Communities smaller than $\sqrt{L}$ nodes may be incorrectly merged
- **Imbalanced communities**: Tends to produce some very large communities alongside many small ones
- **Degeneracy**: Multiple different partitions can have nearly identical modularity values
  - Different runs or small perturbations can yield substantially different communities

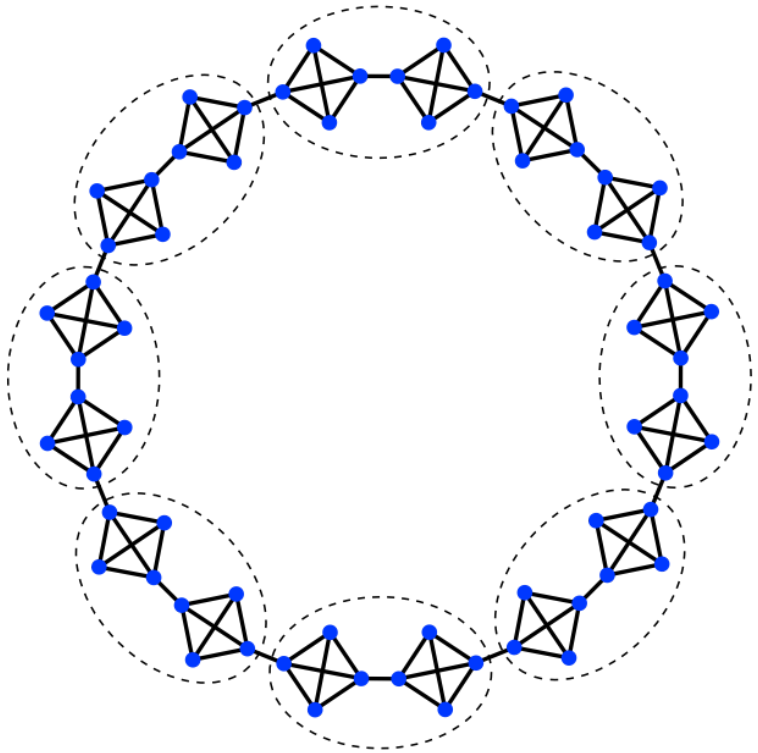
Modularity optimization: limits

- The maximum modularity tends to be larger on larger networks, so **the measure cannot be used to compare the quality of partitions across networks**
- 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.
- Small communities may not be detected **(resolution limit)**
- Reading material (**very useful** if you want to explore the modularity concept in detail):
  - Resolution limit in community detection. Santo Fortunato and Marc Barthélemy

Modularity limits: Resolution limit explained

- **Fundamental issue**: Modularity has an intrinsic scale that depends on network size
- **Mathematical boundary**: Communities smaller than $\sqrt{L}$ links are difficult to detect
- **Intuitive explanation**:
  - Modularity compares actual vs. expected edges
  - For small communities with ~k nodes, they have ~k² possible internal edges
  - When k²/L becomes too small, statistical fluctuations in the random model overwhelm the signal
  - This happens when community size k approaches $\sqrt{L}$
- **Root cause**: The null model assumes a global mixing of links, which creates an artificial "resolution" below which communities become statistically indistinguishable

Example: In a ring of cliques, modularity is higher when small cliques are merged in pairs than when each clique is its own community



Modularity limits: Degeneracy problem

- **Multiple solutions**: For complex networks, many different partitions can have nearly identical modularity values
- **Flat landscape**: The modularity function has a large plateau near the optimal value
- **Practical impact**: Different algorithms or even different runs of the same algorithm may produce drastically different partitions with similar modularity scores
- **Statistical significance**: When many equally good solutions exist, the meaning of any single solution becomes questionable
- **Visualization**: The space of partitions resembles a mountainous landscape with many peaks of similar height

Louvain Algorithm

The Louvain algorithm is a fast, hierarchical community detection method that optimizes modularity through two alternating phases.

Phase 1: Local Modularity Optimization

1. **Initialization**:

- Begin with each node in its own community (N communities)
- Each node is labeled with a unique community ID

2. **Local Moving Phase**:

- For each node i, in random order:
  - Remove i from its current community
  - Place i in the neighboring community that maximizes modularity gain
  - If no modularity gain is possible, keep i in its original community

Phase 2: Network Aggregation

1. **Build a Supernetwork**:

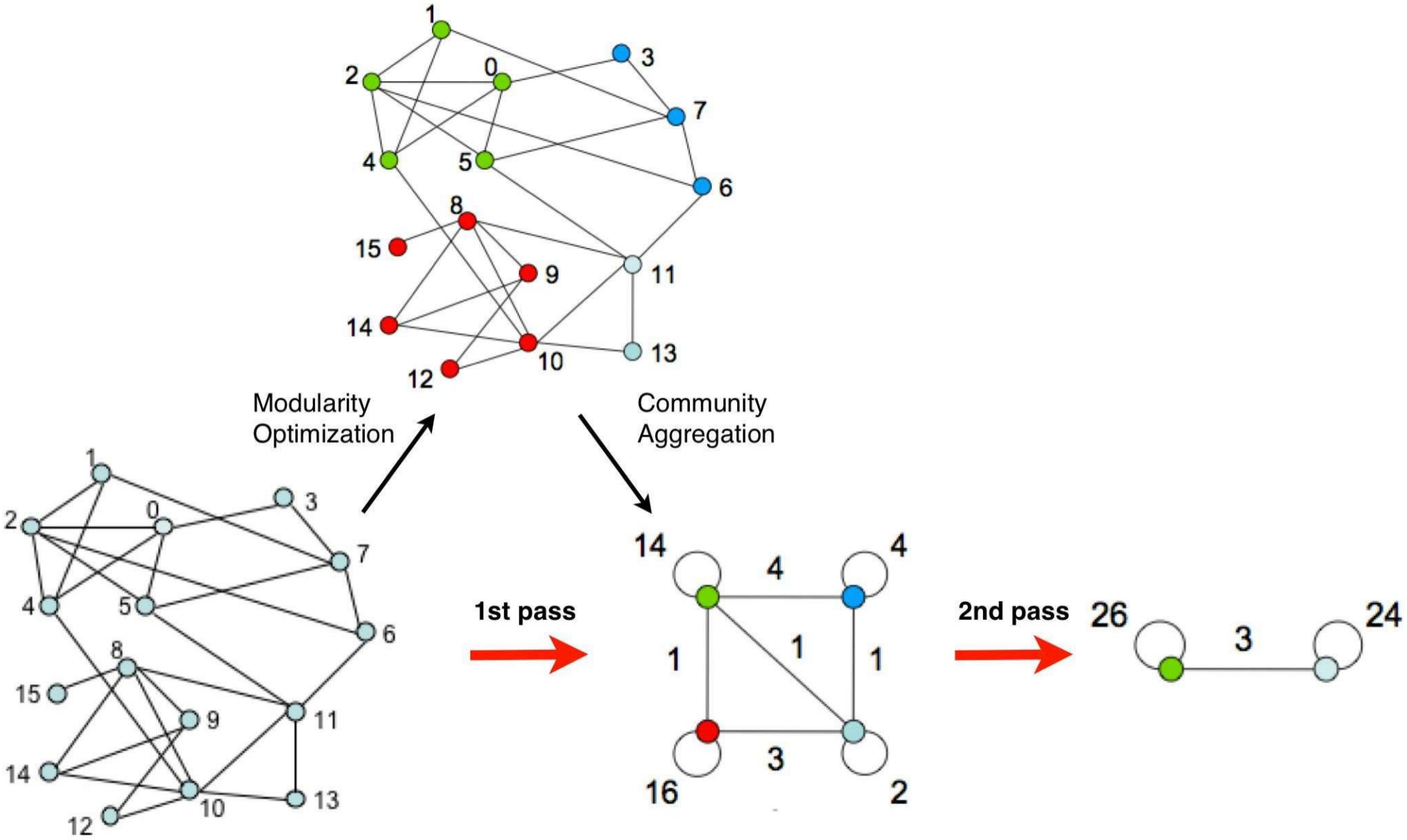
- Each community becomes a "supernode"
- Create weighted edges between supernodes:
  - Weight = sum of weights of all edges between original communities
- Self-loops represent internal connections within communities

2. **Iteration**:

- Apply Phase 1 to this new supernetwork
- Communities of supernodes = communities of communities

Termination

- Alternate Phase 1 and Phase 2 until no further modularity improvement is possible
- The final partition is a hierarchy of communities at different resolutions


Modularity
Optimization
Community
Aggregation
1st pass
2nd pass
14
4
4
1
1
1
3
16
2
26
3
24

Louvain's algorithm

- Note that **modularity is always computed with respect to the original network** since we ultimately care about clustering the actual nodes and not the supernodes

In NetworkX:

```
# Find the best partition of a graph using the Louvain Community Detection Algorithm
louvain_communities(G, weight='weight', resolution=1, threshold=1e-07)
```

- Reading material:
  - V.D. Blondel, J.L. Guillaume, R. Lambiotte, and E. Lefebvre, Fast unfolding of communities in large networks, in Journal of Statistical Mechanics: Theory and Experiment, Volume 2008, October 2008

Louvain's algorithm: Advantages

- **Computational efficiency**: Can analyze networks with millions of nodes in reasonable time
- **Hierarchical detection**: Naturally reveals multilevel community structure
- **Quality results**: Generally produces partitions with high modularity
- **Flexibility**: Works with weighted, directed, and multi-graphs
- **No predefined community number**: Automatically determines optimal number of communities

Louvain's algorithm: Limitations

- **Resolution limit**: Inherits modularity's inability to detect small communities
- **Degeneracy**: Multiple different partitions may have similar modularity scores
- **Order dependency**: Results can vary based on the order of node processing
- **Instability**: Small network changes can sometimes lead to substantially different partitions
- **Local optimum**: As a greedy method, may not find global modularity maximum

Label propagation

Principle: neighbors of a node usually belong to the same community

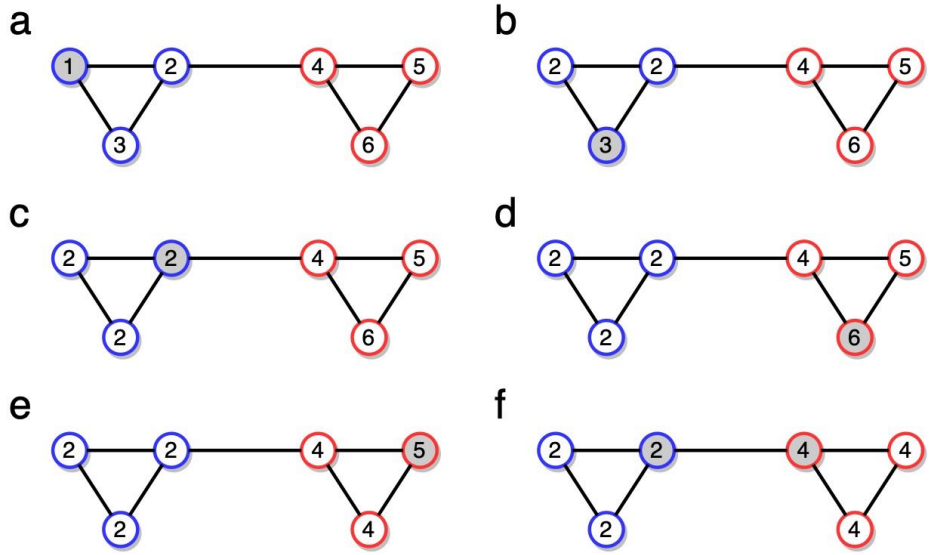
Procedure:

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 equal 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

- The algorithm **does not deliver a unique solution**; the outcome depends on the order in which the nodes are visited in each sweep
- 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
- Despite these instabilities, the partitions found by label propagation in real networks **tend to be similar to each other**. For more robust results, one can combine the solutions obtained from different runs of the procedure
- Reading material:
  - Raghavan, Usha Nandini, Réka Albert, and Soundar Kumara. "Near linear time algorithm to detect community structures in large-scale networks." Physical Review E 76.3 (2007): 036106

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

In NetworkX:

```
partition = nx.community.asyn_lpa_communities(G)
```

Stochastic block modeling

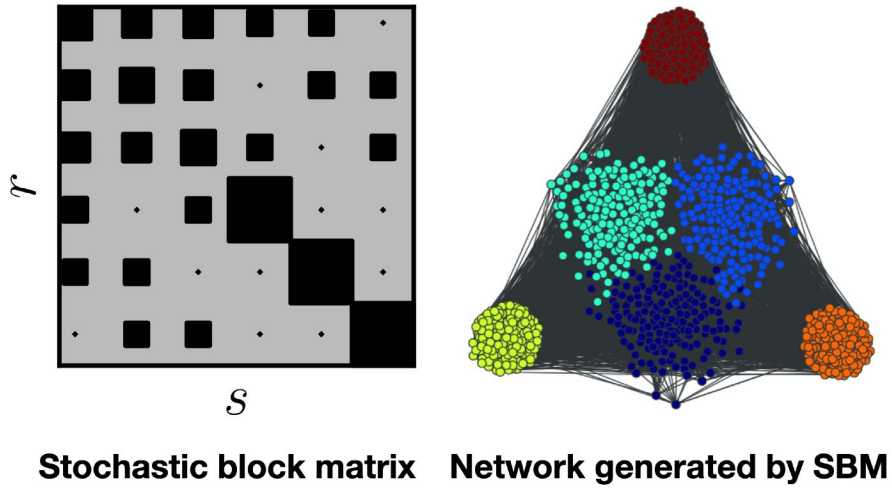
- **Basic assumption:** the network has been generated by some model, e.g., one of the models we have seen in the class on Network Models
- Models are characterized by one or more parameters
- **Question:** for which parameter values does the model produce networks most closely resemble the one we are 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, node i is in group g_i
- The probability to form a link between nodes i and j depends exclusively on their group memberships and is $P(i \leftrightarrow j) = p_{g_i g_j}$
- For any pair of groups g_1 and g_2, the connection probability between any node in g_1 and any node in g_2 is identical.
- The **stochastic block matrix** is the $q \times q$ matrix whose element kl is the link probability $p_{g_k g_l}$
- 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

- 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 **disassortative structure**
  - links are more likely between than within blocks
- In the special case of $p_{rr} = 0$ we have **mutipartite networks**
  - links only between the 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 all probabilities are equal $p_{rs} = p \quad \forall r, s = 1, \ldots, q$ we recover the **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, so that $P(i \leftrightarrow j) = p_{g_i g_j} c_i c_j / 2L$ where c_i is the desired average degree of node i

Fitting a stochastic block model: intuitive approach

- **Concept:** Fitting a SBM is like "reverse engineering" - we're trying to find the community structure that most likely generated our observed network
- **Intuitive process:**
  i. We start with a network and a guess about its communities
  ii. Ask: "If these were the true communities, what would the connection probabilities be?"
  iii. Compare: "How well does this explain the actual connections we observe?"
  iv. Iterate: Try different community assignments until we find the best explanation

Fitting a stochastic block model: the mathematics

The probability that a network G is reproduced by the DCSBM based on a given partition g of G's nodes into q groups is expressed by the **log-likelihood**

$$\mathcal{L}(G|g) = \sum_{r,s=1}^{q} L_{rs} \log\left(\frac{L_{rs}}{k_r k_s}\right)$$

Where:

- L_{rs} = number of links between group r and group s
- k_r = sum of degrees of all nodes in group r
- $\frac{L_{rs}}{k_r k_s}$ = ratio of actual vs. expected connections between groups

Interpretation: This formula rewards partitions where groups have more internal connections than would be expected by chance, accounting for the degrees of nodes.

Stochastic block model fitting: practical algorithm

Step-by-step procedure:

1. **Initialization**:
   - Start with a random assignment of nodes to q groups
   - Calculate initial log-likelihood score
2. **Optimization**:
   - For each node in random order:
     - Temporarily move it to each possible group
     - Calculate the change in log-likelihood for each move
     - Place the node in the group that gives highest improvement
   - Repeat until no further improvement is possible
3. **Multiple restarts**:
   - Run the algorithm several times with different initial conditions
   - Select the partition with the highest overall likelihood

Reading material

References

[ns1] **Chapter 6 : Communities**

Q&A