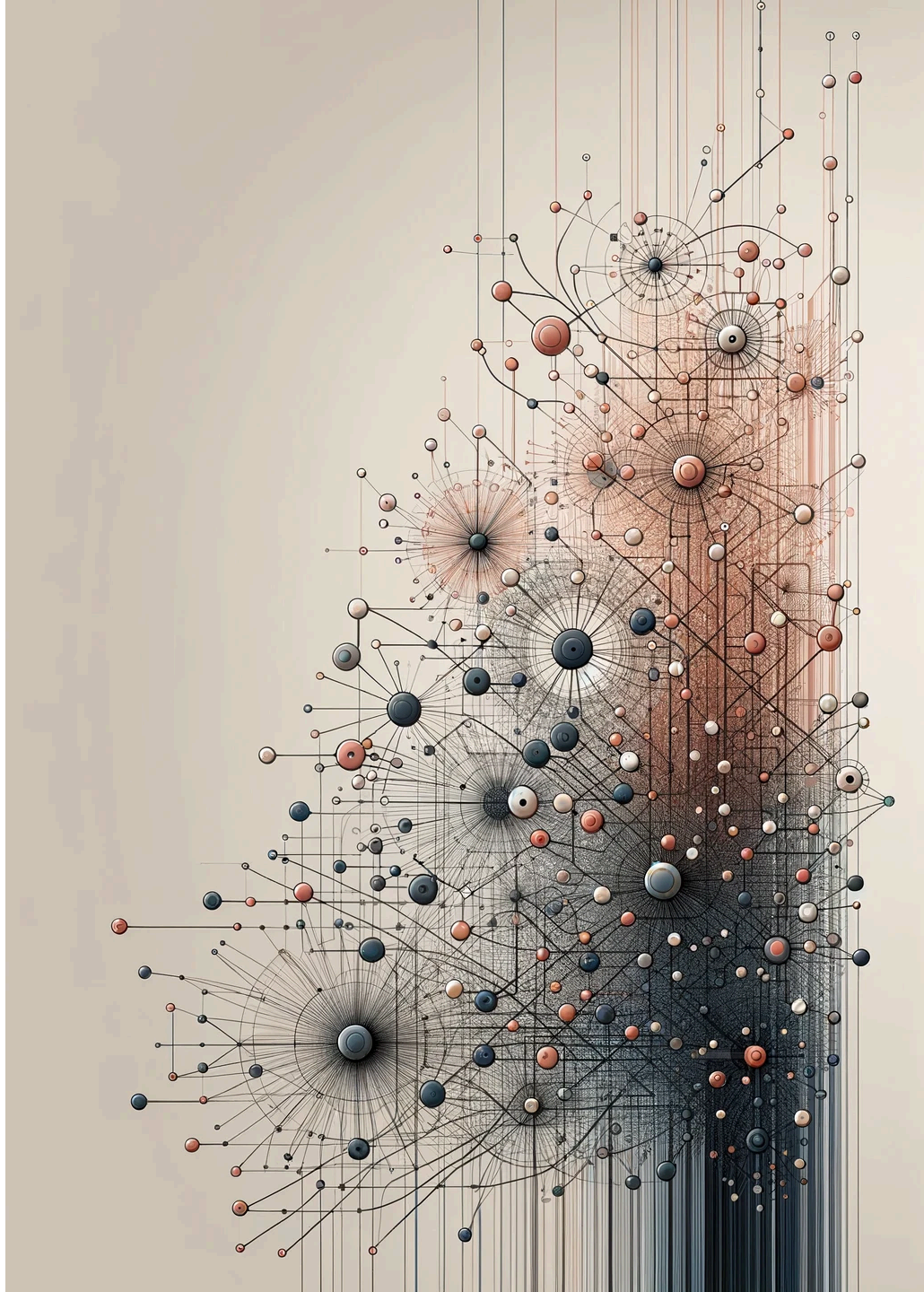
UNIVERSITÀ DI TORINO

Analisi e Visualizzazione delle Reti Complesse

NS12 - Community Detection

Prof. Rossano Schifanella



UNIVERSITÀ DI TORINO

Outline

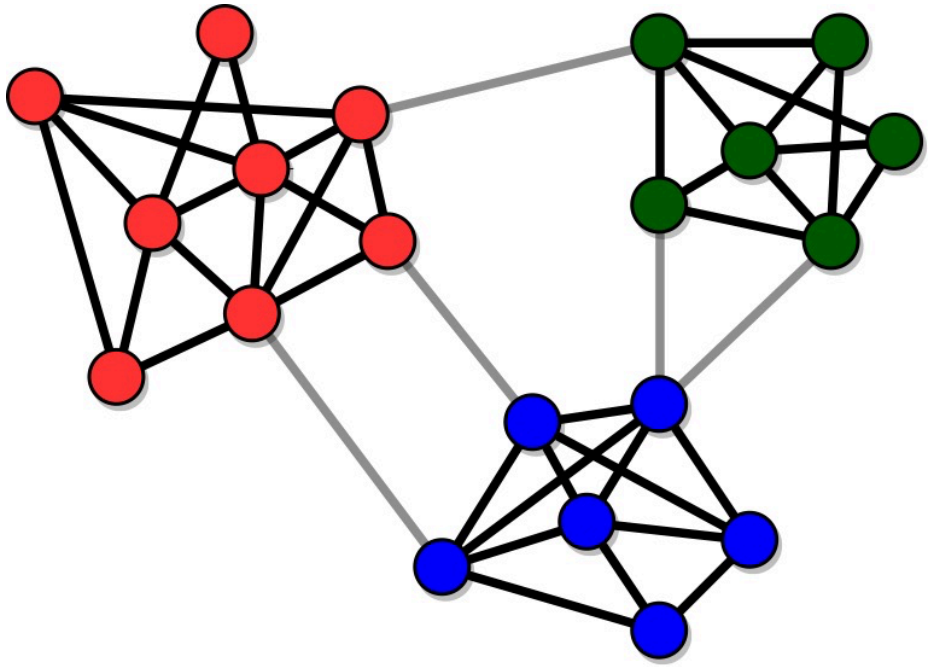
- Community detection
  - Girvan-Newman
  - Modularity and modularity optimization
  - Newman's Greedy Algorithm
  - Louvain Algorithm
  - Modularity's limits
  - Label propagation
  - Stochastic block modeling

UNIVERSITÀ DI TORINO

Community detection

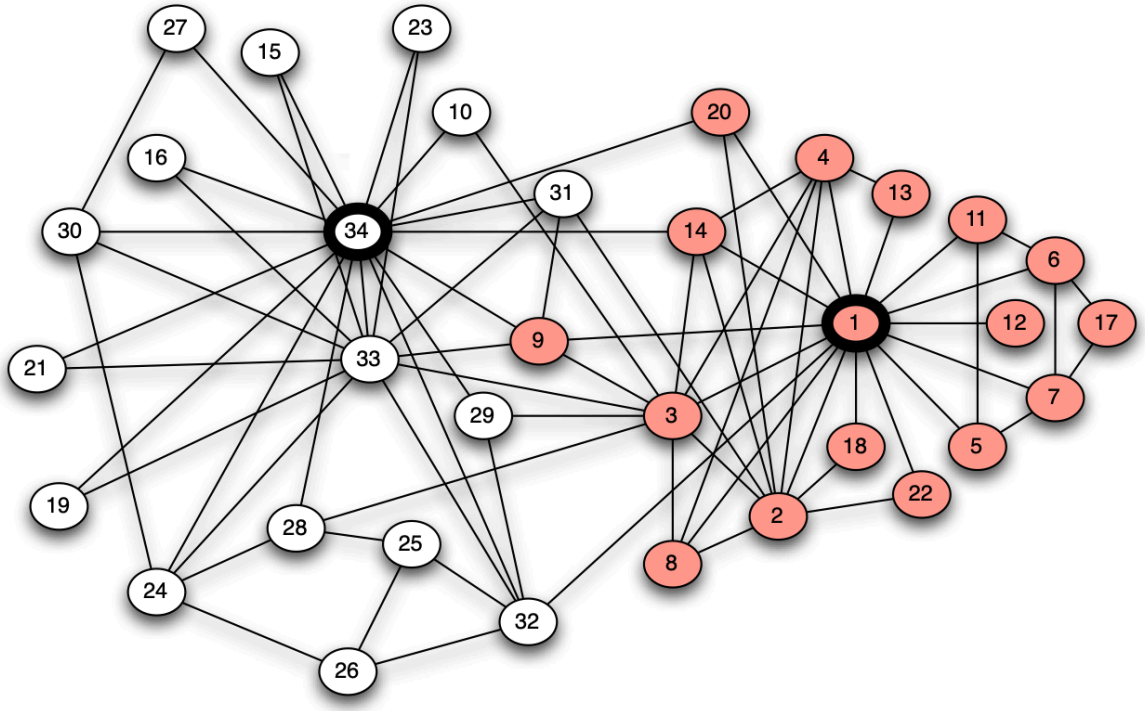
Community detection

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



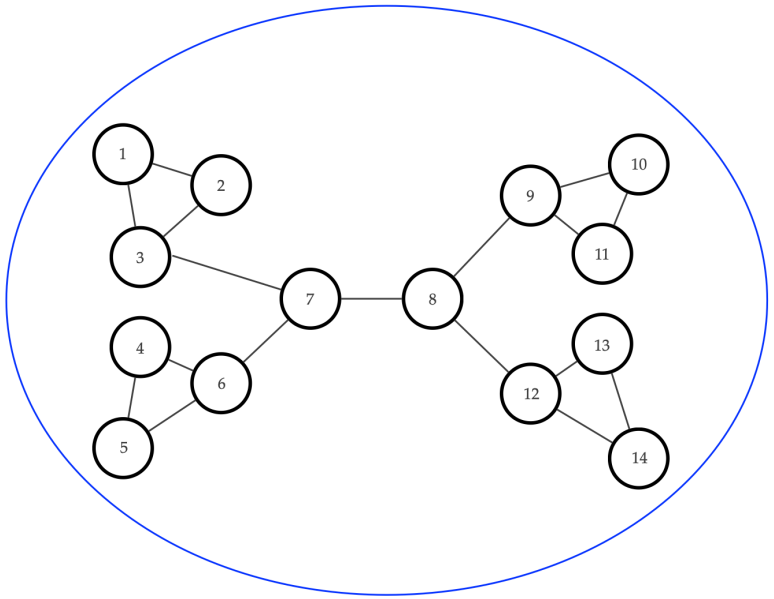
Example: Zachary Club

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



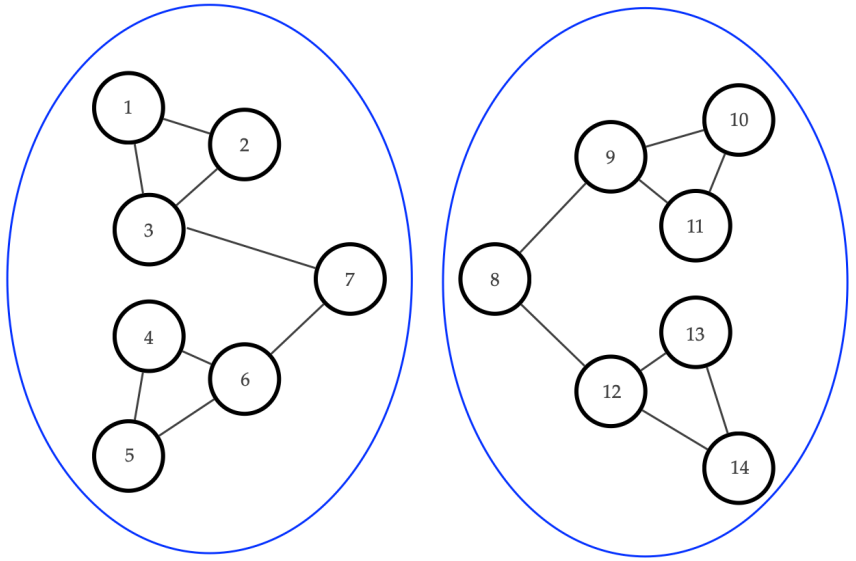
General approaches

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



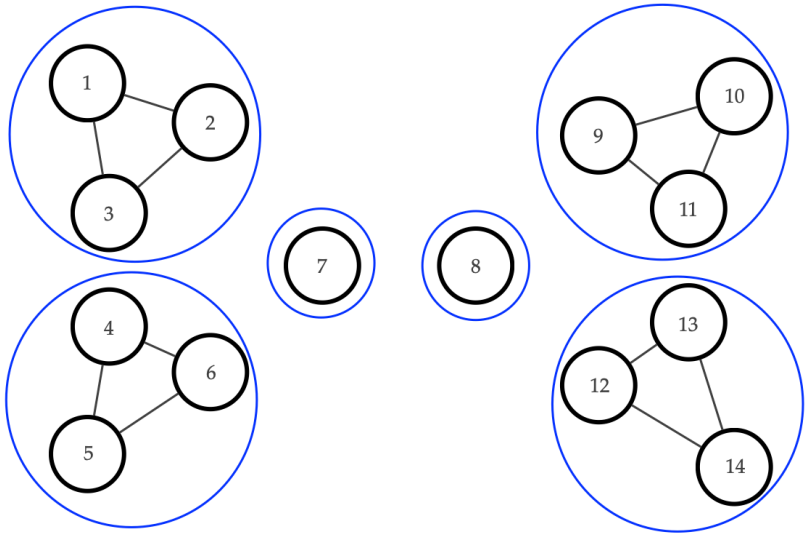
General approaches

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



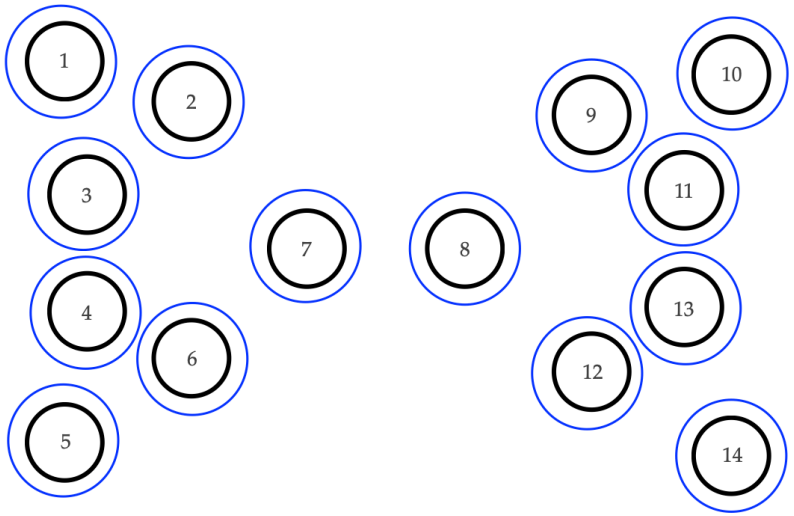
General approaches

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



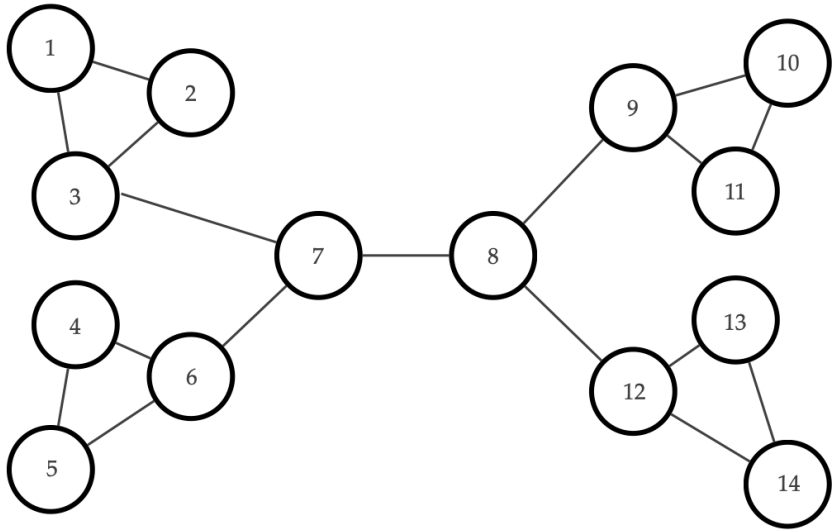
General approaches

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



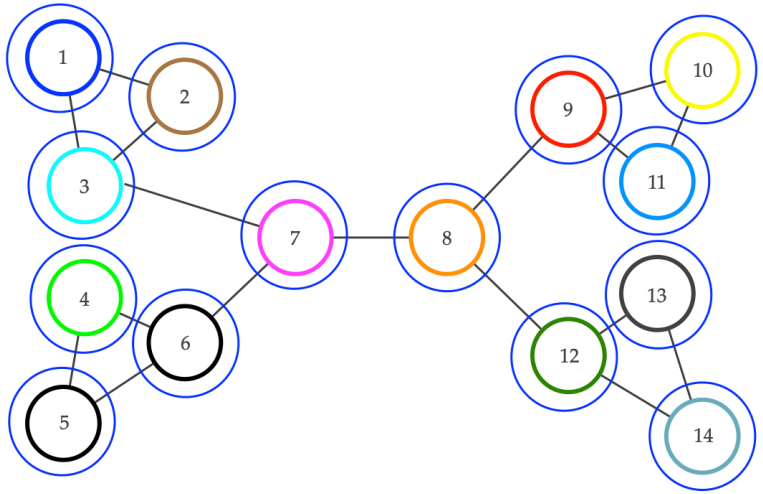
General approaches

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



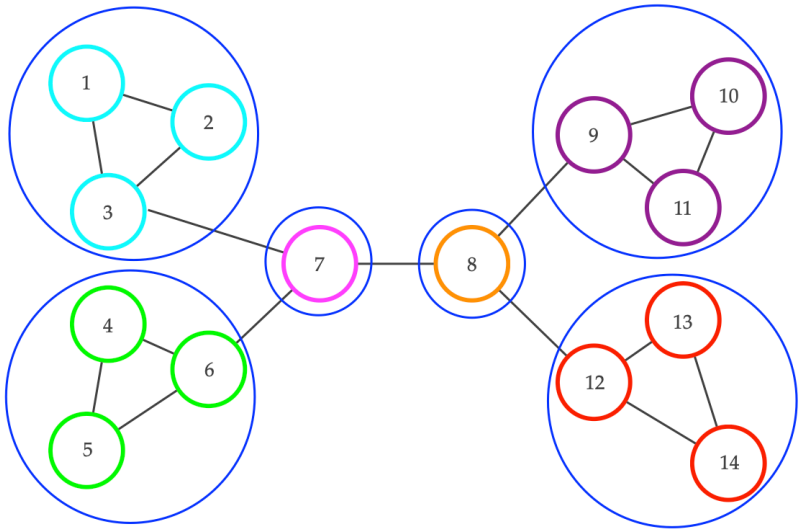
General approaches

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



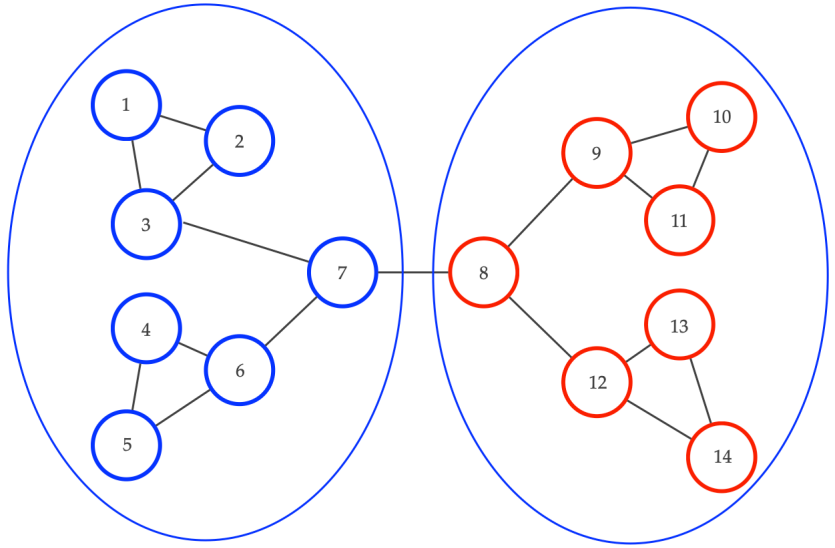
General approaches

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



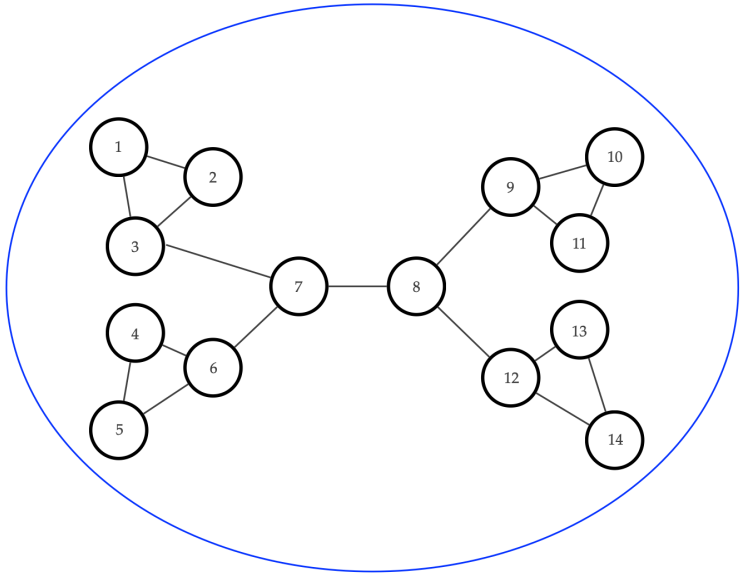
General approaches

- **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 $\rightarrow \infty$

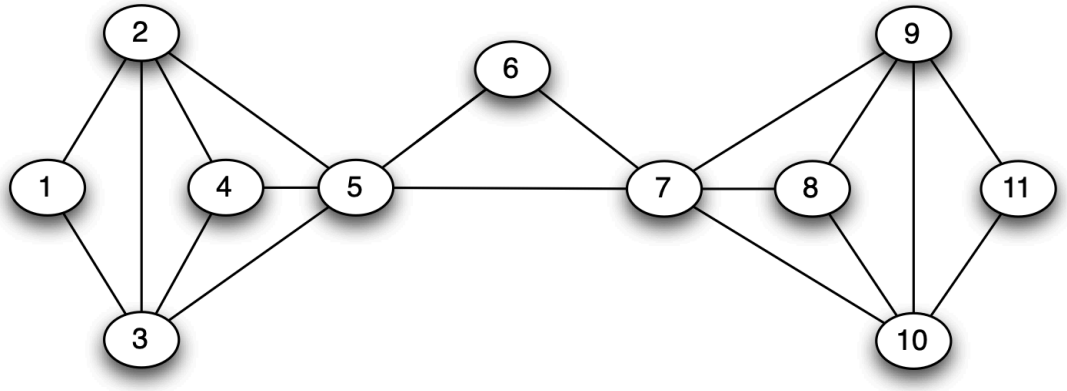
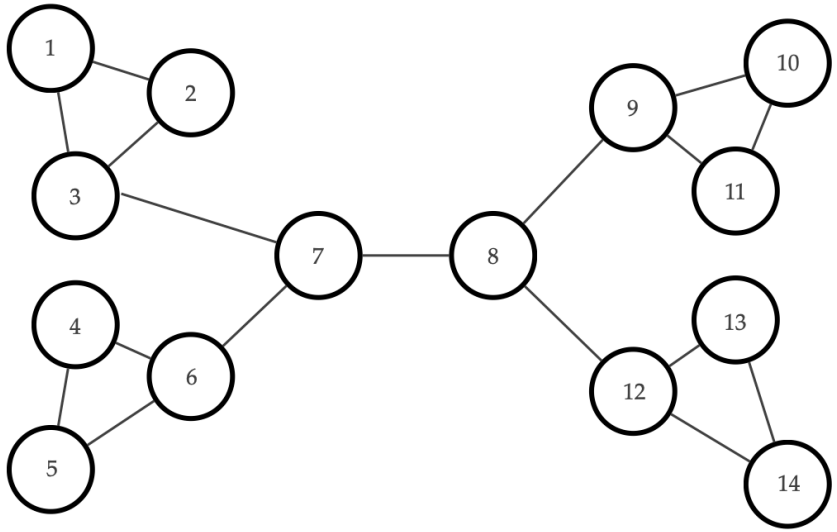


Figure 3.15: A network can display tightly-knit regions even when there are no bridges or local bridges along which to separate it.

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**

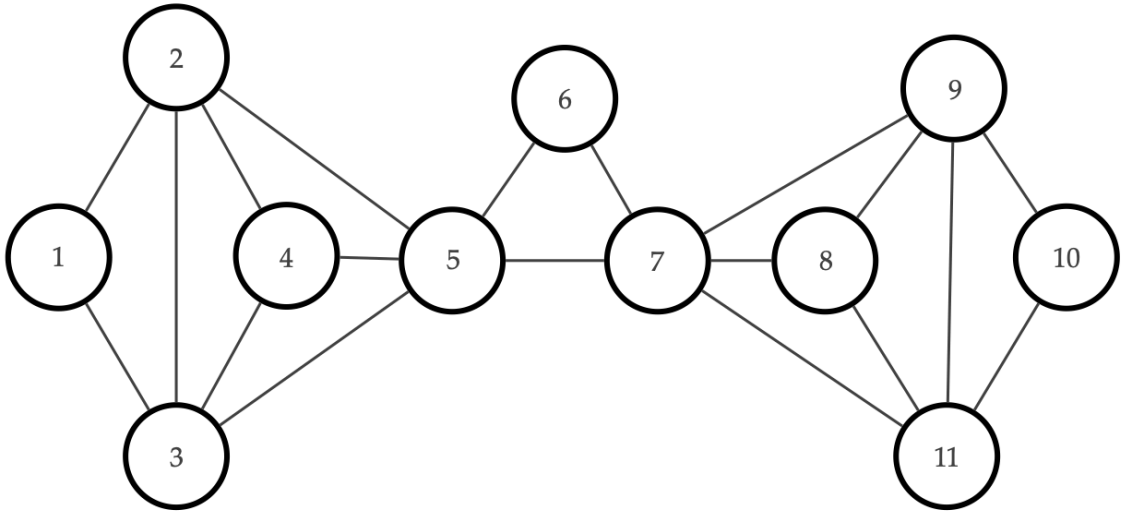
The Girvan-Newman method

Procedure:

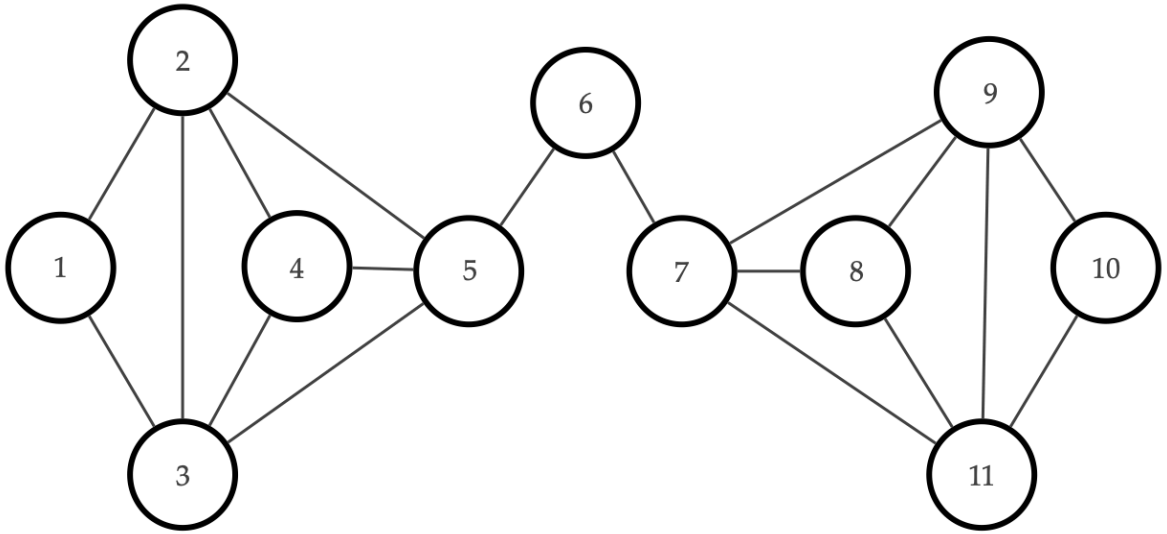
1. Calculate betweenness for all the edges
2. Find edge(s) with highest betweenness
3. Remove those edges
4. **if** satisfied **or** no more edges **return** components as clusters
5. **else** goto 1.

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

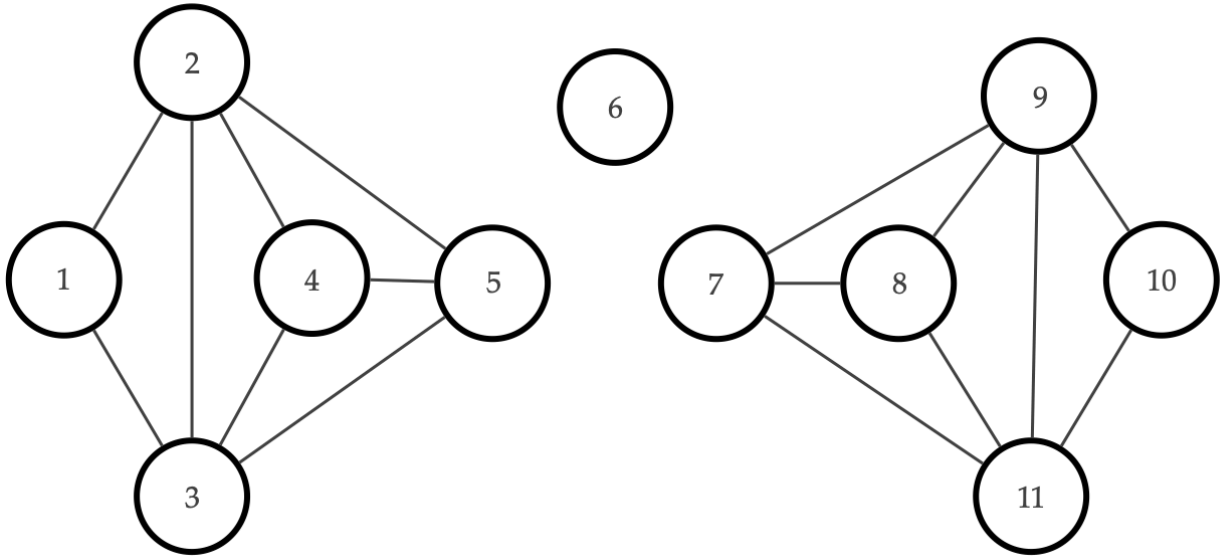
The Girvan-Newman method



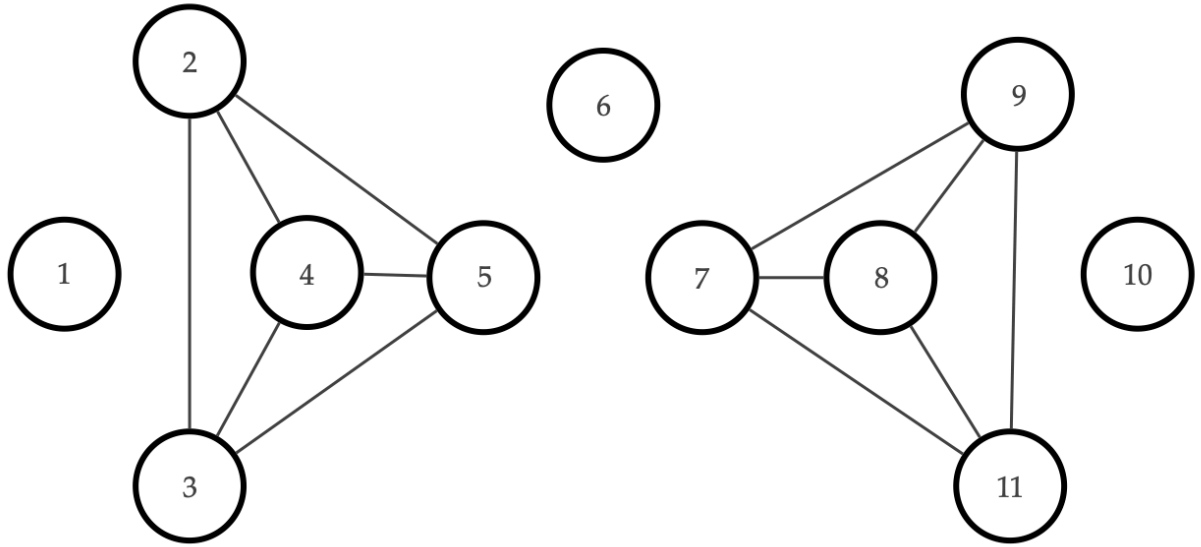
The Girvan-Newman method



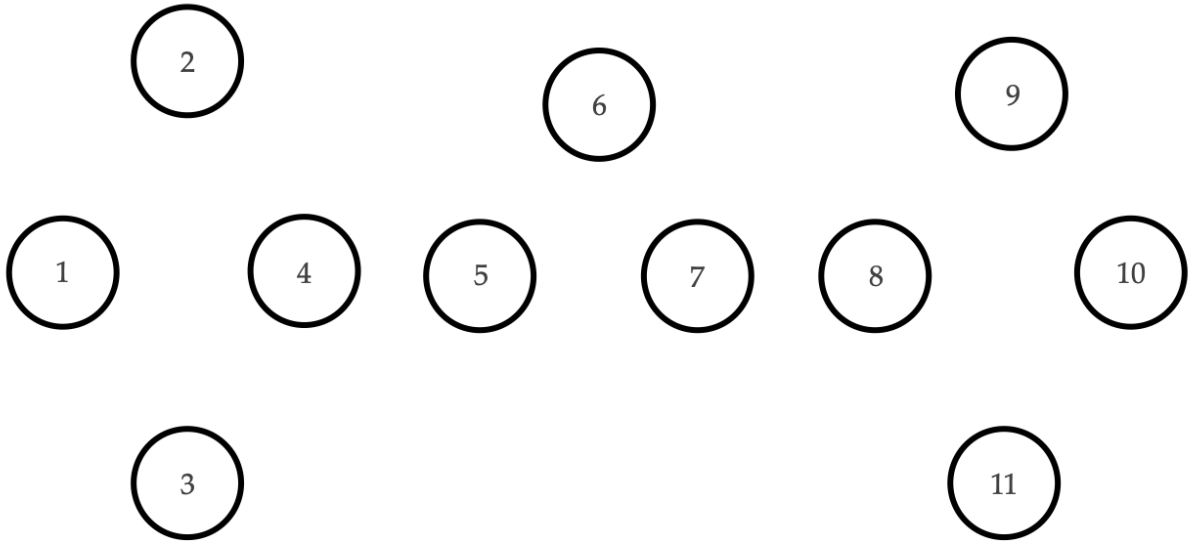
The Girvan-Newman method



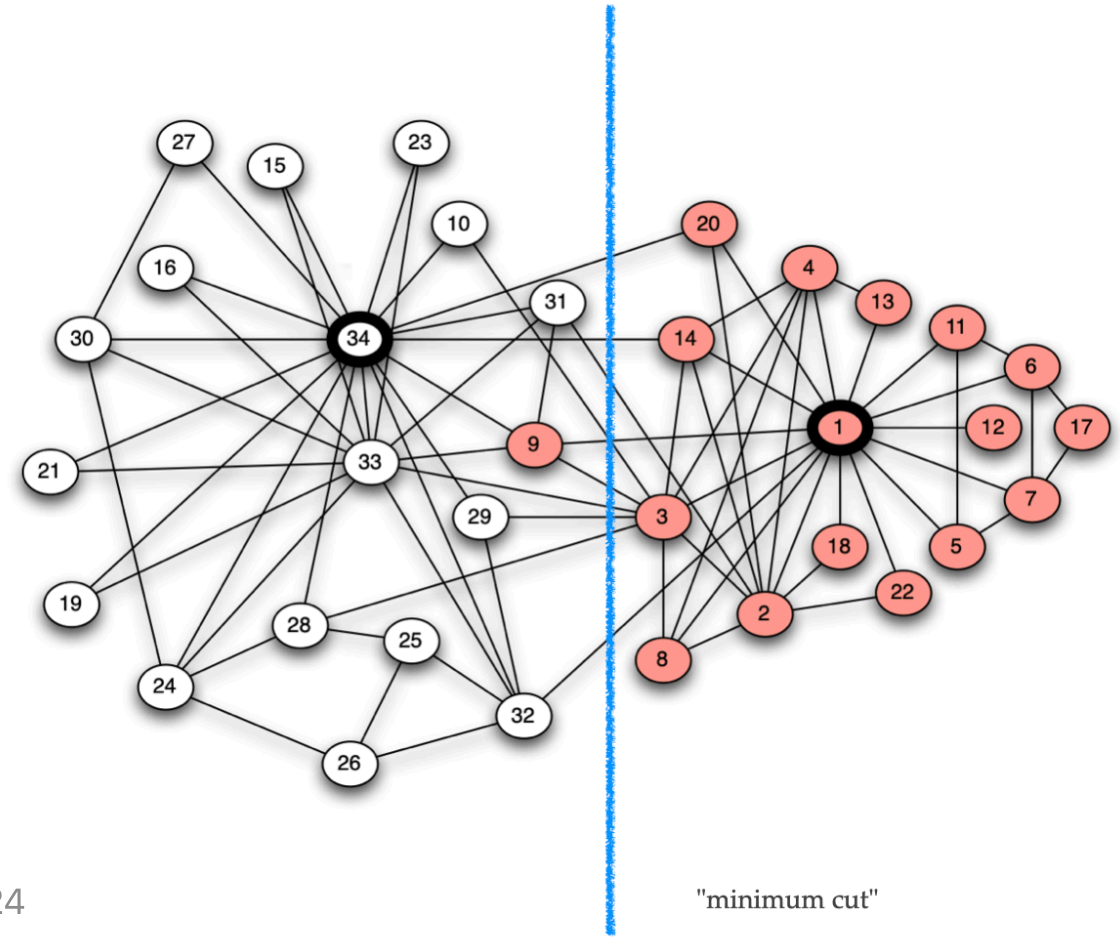
The Girvan-Newman method



The Girvan-Newman method



Applying Girvan-Newman to Zachary Club



Observations and questions on Girvan-Newman

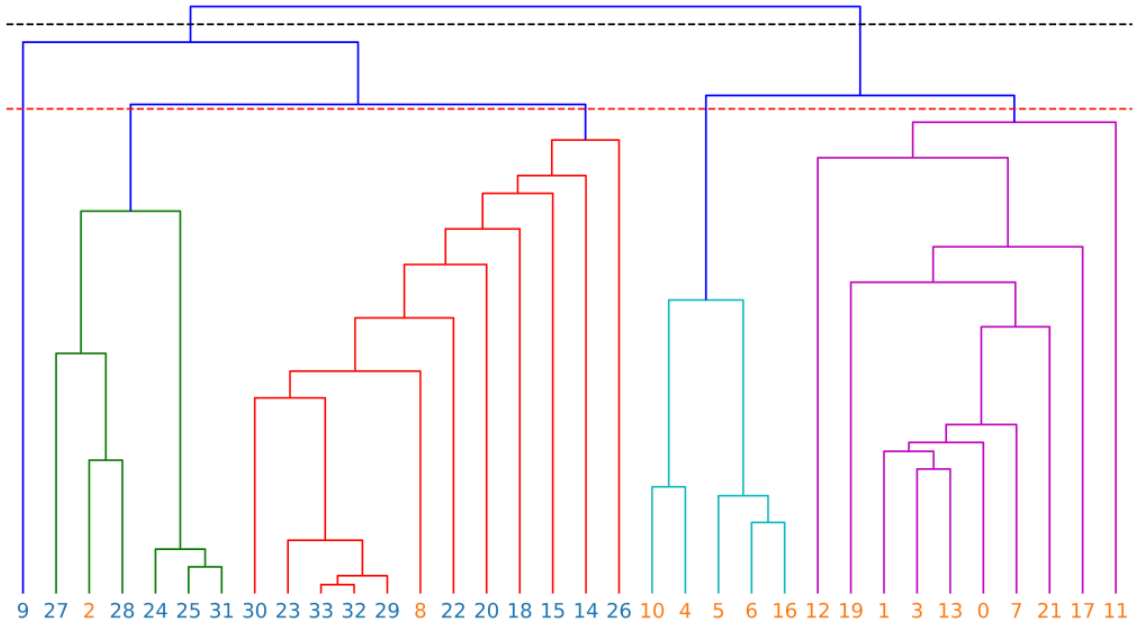
- We do not need to find **local** bridges
- betweenness is an excellent approximation that is calculated through a **global** property of the network
- How to efficiently calculate edge's betweenness?
  - see the previous class!
- When do I stop?
  - many criteria, more later

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 with the one we get from the Configuration Model

Modularity

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

L : number of links in the network

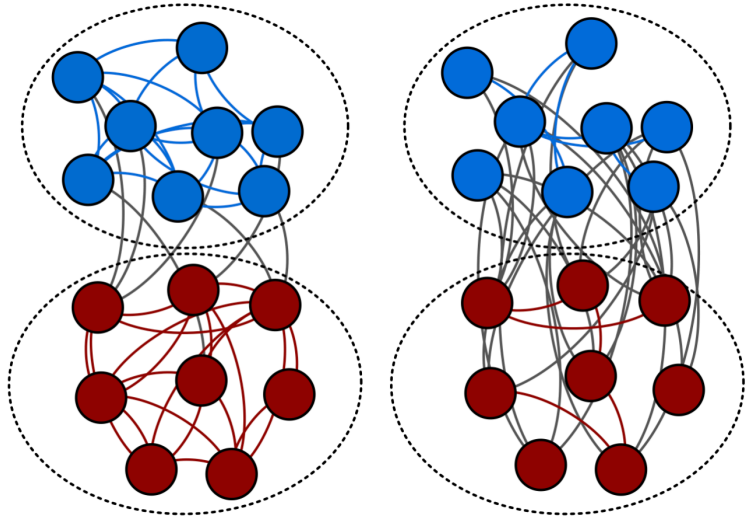
L_C : number of internal links in community C

k_C : degree of community C

$\frac{k_C^2}{4L}$: expected number of internal links in community C

Modularity

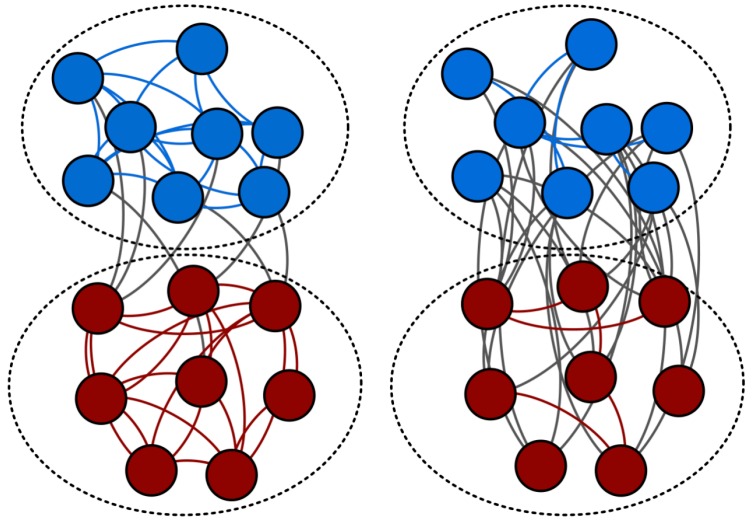
- Let's explain the origin of $\frac{k_C^2}{4L}$
- Random links are formed by matching pairs of stubs (half-links) chosen at random
- The total number of stubs attached to C is k_C
- The probability to select one of those stubs at random is $\frac{k_C}{2L}$ because $2L$ is the total number of stubs of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



Original network Randomized network

Modularity

- 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: $p_C = (\frac{k_C}{2L} \cdot \frac{k_C}{2L} = \frac{k_C^2}{4L})$
- 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 $\frac{k_C^2}{4L}$



Original network **Randomized network**

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

In NetworkX:

```
# 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 in community C

k_C^{in} : total in-degree of nodes in community C

k_C^{out} : 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 the community C

s_C : total strength of nodes in the community C, i.e., the 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 the community C

s_C^{in} : total in-strength of nodes in community C

s_C^{in} : 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 the largest modularity

In NetworkX:

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

Newman's greedy algorithm: limits

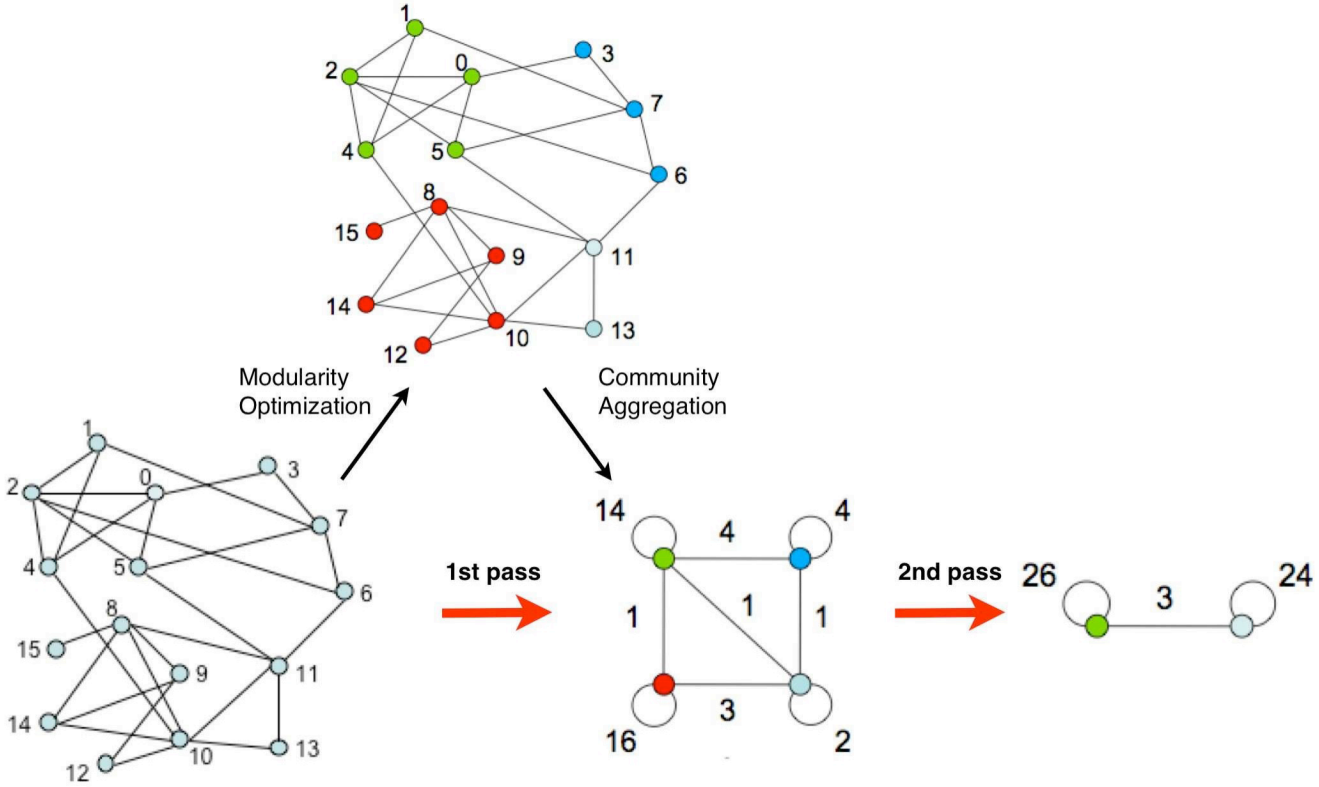
- The method is greedy, in that it tries to maximize the modularity at each step. As such, it is likely to get stuck on solutions with sub-optimal modularity
- It tends to generate unbalanced partitions, with some clusters much larger than the others. Because of that, the method is not very fast
- Merging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

Louvain's algorithm

Procedure:

1. **Start:** each node is assigned to a different community
2. **Loop over the nodes:** each node is put in the community of the neighbor that yields the largest modularity increase ΔQ with respect to the current partition. All nodes are revisited over and over until it is no longer possible to increase Q by moving a node to a different community
3. The network is transformed into a **weighted supernetwork**, where each community in the partition from step 2 is replaced by a supernode, links between supernodes are weighted by the number of links joining nodes in the corresponding groups, and links joining nodes in the same community are represented as a self-loop from the corresponding supernode to itself, with weight equal to the number of internal links
4. The procedure stops when no further grouping of the clusters in the current partition increases the modularity

Louvain's algorithm



- 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

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

Advantages:

- 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's algorithm

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)
```

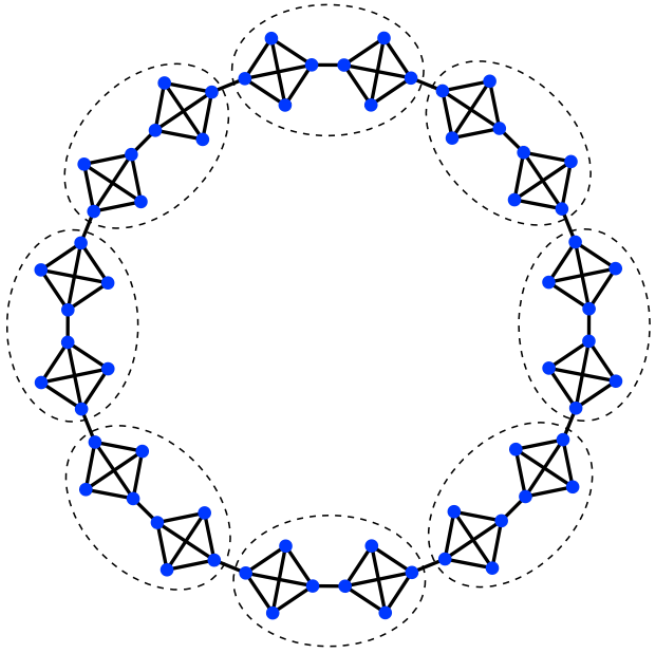
Modularity optimization: limits

- 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. Communities smaller than a certain size may not be detected **(resolution limit)**

Modularity: resolution limit

- 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

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



Hierarchical clustering: limits

- It delivers as many partitions as nodes: which one(s) shall we choose?
- Results usually depend on the similarity measure and on the criterion adopted to compute the similarity of the groups
- It is rather slow; networks with millions of nodes are out of reach

Label propagation

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

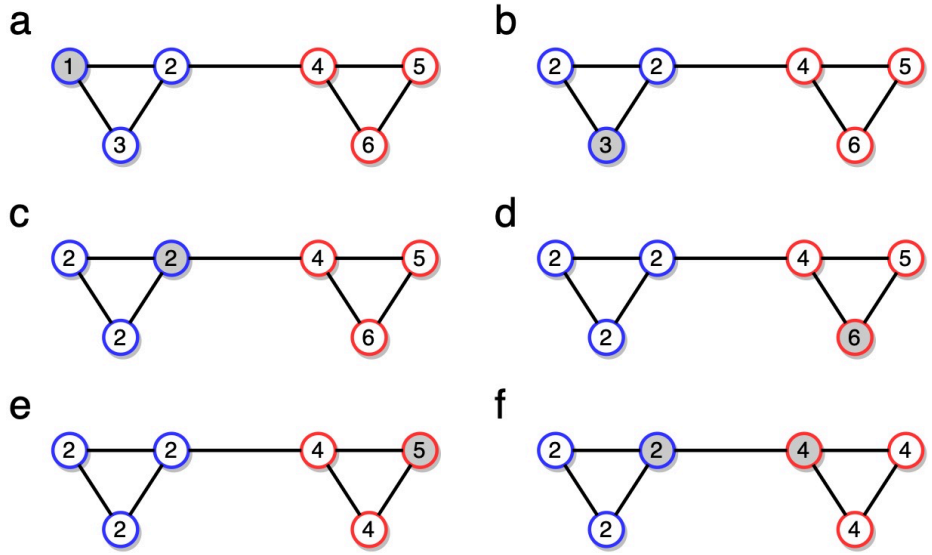
Procedure:

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

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:

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

Stochastic block modeling

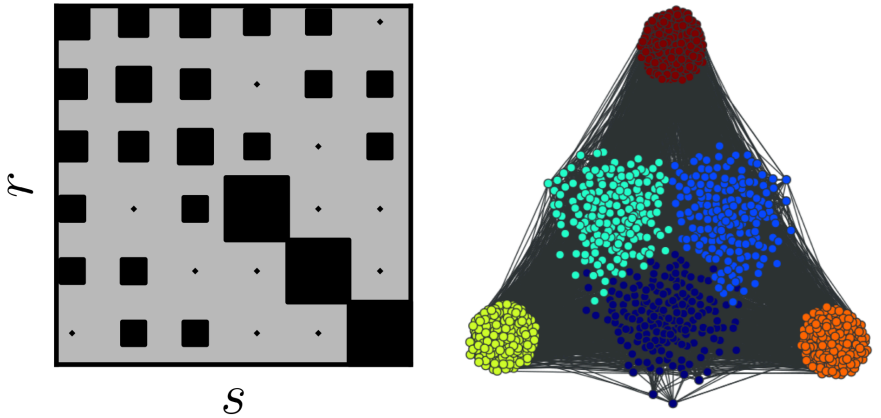
- **Basic assumption:** the network has been generated by some model, e.g., one of the models we have seen in Chapter 5
- Models are characterized by one or more parameters
- **Question:** for which parameter values does the model produce networks most closely resemble the one we are investigating?
- **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, 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

- If $\forall r, s = 1, \dots, q$ with $r \neq s$ we have $p_{rr} > p_{rs}$, we have **community structure**
  - links are more likely within than between blocks
- If $\forall r, s = 1, \dots, q$ 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 $p_{rs} = p \quad \forall r, s = 1, \dots, q$ 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

Fitting a stochastic block model to a network

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**

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

L_{rs} : number of links between group r and group s

k_r : degree of group r (sum of degrees of nodes in r)

Fitting a stochastic block model to a network

Procedure:

1. Start: random partition in q clusters
2. Repeatedly move a node from one group to another, selecting at each step the move that will most increase the likelihood (or least decrease it, if no increase is possible), under the constraint that each node may be moved only once
3. When all nodes have been moved, inspect the partitions through which the system passed from start to end of the procedure in step 2 and select the one with the highest likelihood
4. Stopping criterion: the likelihood for two consecutive iterations is the same, i.e., it cannot be increased any further

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

Reading material

References

[ns1] **Chapter 6 : Communities**

[ns2] **Chapter 3: (3.6-3.6.A)**

UNIVERSITÀ DI TORINO

Q&A