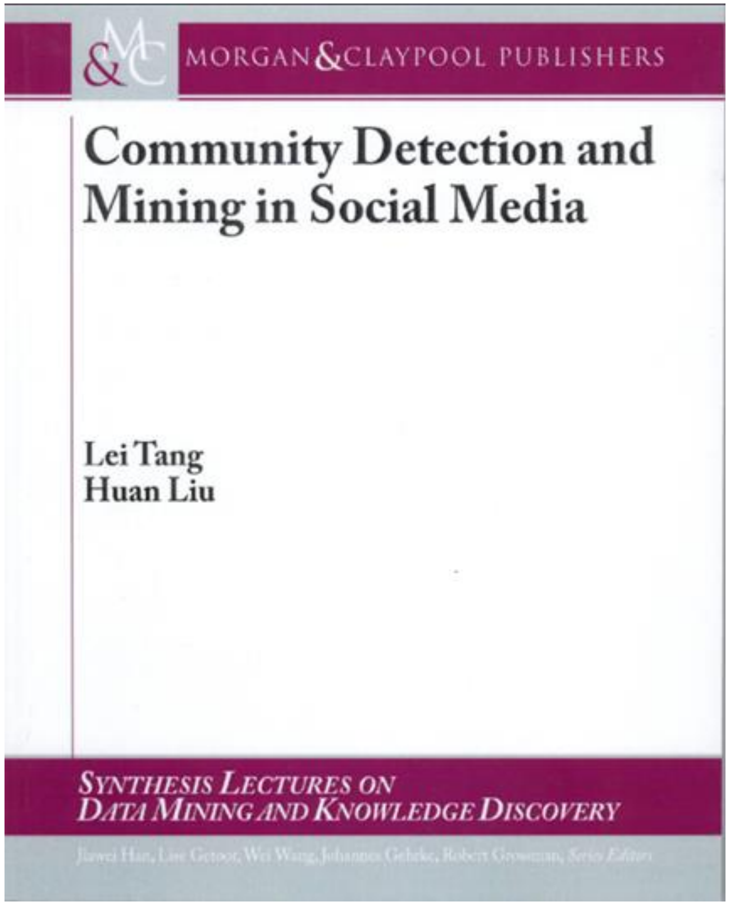
GRAPH PARTITIONING AND COMMUNITY DETECTION



MORGAN&CLAYPOOL PUBLISHERS

Community Detection and Mining in Social Media

Lei Tang
Huan Liu

Synthesis Lectures on Data Mining and Knowledge Discovery

Jiawei Han, Lise Getoor, Wei Wang, Johannes Gehrke, Robert Grossman, *Series Editors*

Slides based on chapter 3 of the Tang and Liu book

Group centric

- A group-centric criterion considers connections inside a group as whole. It is acceptable to have some nodes in the group to have low connectivity as long as the group overall satisfies certain requirements.

- One such example is density-based groups

- a complete subgraph $H(V_H, E_H)$ of $G(V, E)$ where $|V_H|=n_H$ and $|E_H|=m_H$ is γ-dense are (also called a quasi-clique) if:

$$\frac{m_H}{n_H(n_H - 1)/2} \geq \gamma$$

- Clearly, the quasi-clique becomes a clique when γ=1

- Note that this density-based group-centric criterion does not guarantee reachability for each node in the group. It allows the degree of a node to vary, thus is more suitable for large-scale networks.

Group centric

- Not a trivial task to search for quasi-cliques. Strategies similar to those of finding cliques
- Abello et al. (2002): iterative procedure consists of two steps

• Local search: Sample a sub-network and search for a maximal quasiclique in it. A greedy approach is to expand a quasi-clique by encompassing those high-degree neighboring nodes until the density drops below γ .

• Heuristic pruning: If we know a γ -dense quasi-clique of size k, then a heuristic is to prune those "peelable" nodes and their incident edges. A node v is peelable if v and its neighbors all have degree less than $k\gamma$ because it is less likely to contribute to a larger quasi-clique. We can start from low-degree nodes and recursively remove peelable nodes in the original network.

This process is repeated until the network is reduced to a reasonable size so that a maximal quasiclique can be found directly.

Network centric

Network-centric community detection has to consider the global topology of a network.

It aims to *partition* nodes of a network into a number of disjoint sets.

Typically (but not always), network-centric community detection aims to optimize a criterion defined over a network partition rather than over one group.

A group in this case is not defined independently.

Toy example

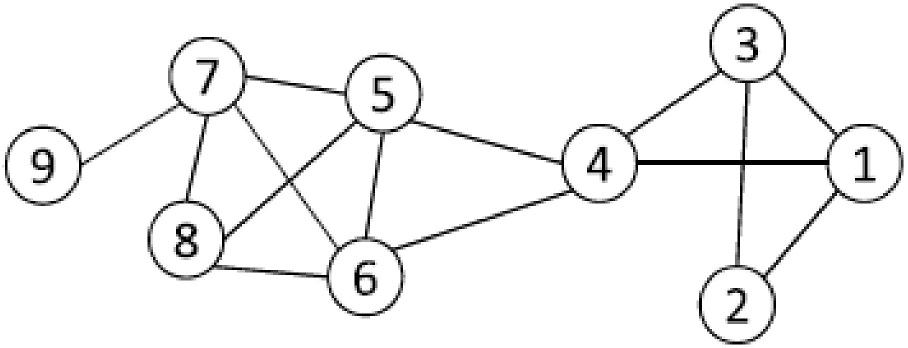


Figure 1.1: A social network of 9 actors and 14 connections. The diameter of the network is 5.

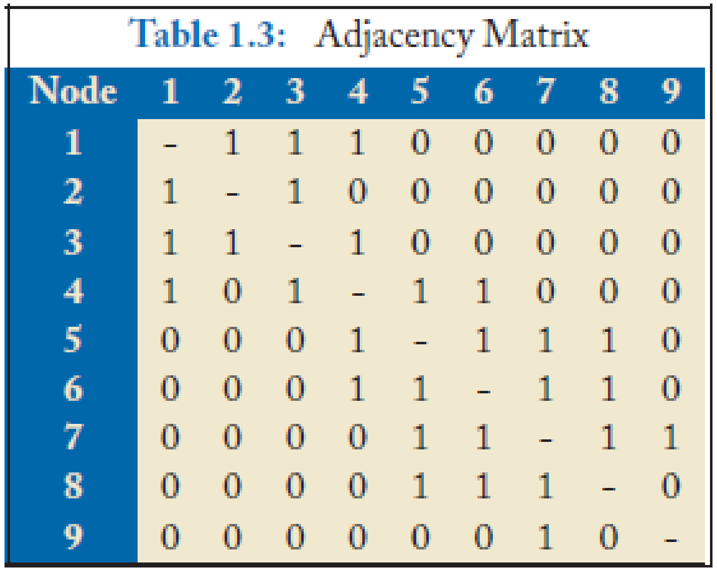


Table 1.3: Adjacency Matrix

Node	1	2	3	4	5	6	7	8	9
1	-	1	1	1	0	0	0	0	0
2	1	-	1	0	0	0	0	0	0
3	1	1	-	1	0	0	0	0	0
4	1	0	1	-	1	1	0	0	0
5	0	0	0	1	-	1	1	1	0
6	0	0	0	1	1	-	1	1	0
7	0	0	0	0	1	1	-	1	1
8	0	0	0	0	1	1	1	-	0
9	0	0	0	0	0	0	1	0	-

Latent Space models

- A latent space model maps nodes in a network into a low-dimensional Euclidean space such that the proximity between nodes based on network connectivity are kept in the new space

- then the nodes are clustered in the low-dimensional space using methods like k-means

- One representative approach is multi-dimensional scaling (MDS)

- MDS requires the input of a dissimilarity or distance matrix $P \in R^{n\times n}$, with each entry P_{ij} denoting the distance between a pair of nodes i and j in the network.

Latent Space models

- Let S ∈ $R^{n \times k}$ denote the coordinates of nodes in the k-dimensional space such that S is column orthogonal. It can be shown that

$$SS^T \approx -\frac{1}{2}(I - \frac{1}{n}\mathbf{1}\mathbf{1}^T)(P \circ P)(I - \frac{1}{n}\mathbf{1}\mathbf{1}^T) = \widetilde{P},$$

- where *I* is the identity matrix, 1 an n-dimensional column vector with each entry being 1, and ◦ the element-wise matrix multiplication.

- Suppose V contains the top k eigenvectors of $\tilde{P}$ with largest eigenvalues, Λ is a diagonal matrix of top k eigenvalues = diag($\lambda_1, \lambda_2, \cdots, \lambda_k$). The optimal S is S = V $\Lambda^{1/2}$

- the classical k-means algorithm can be applied to S to find community partitions.

Latent Space models

Take the network in the toy example:

$$P = \begin{bmatrix} 0 & 1 & 1 & 1 & 2 & 2 & 3 & 3 & 4 \\ 1 & 0 & 1 & 2 & 3 & 3 & 4 & 4 & 5 \\ 1 & 1 & 0 & 1 & 2 & 2 & 3 & 3 & 4 \\ 1 & 2 & 1 & 0 & 1 & 1 & 2 & 2 & 3 \\ 2 & 3 & 2 & 1 & 0 & 1 & 1 & 1 & 2 \\ 2 & 3 & 2 & 1 & 1 & 0 & 1 & 1 & 2 \\ 3 & 4 & 3 & 2 & 1 & 1 & 0 & 1 & 1 \\ 3 & 4 & 3 & 2 & 1 & 1 & 1 & 0 & 2 \\ 4 & 5 & 4 & 3 & 2 & 2 & 1 & 2 & 0 \end{bmatrix},$$

Let compute $\widetilde{P}$

$$\widetilde{P} = \begin{bmatrix} 2.46 & 3.96 & 1.96 & 0.85 & -0.65 & -0.65 & -2.21 & -2.04 & -3.65 \\ 3.96 & 6.46 & 3.96 & 1.35 & -1.15 & -1.15 & -3.71 & -3.54 & -6.15 \\ 1.96 & 3.96 & 2.46 & 0.85 & -0.65 & -0.65 & -2.21 & -2.04 & -3.65 \\ 0.85 & 1.35 & 0.85 & 0.23 & -0.27 & -0.27 & -0.82 & -0.65 & -1.27 \\ -0.65 & -1.15 & -0.65 & -0.27 & 0.23 & -0.27 & 0.68 & 0.85 & 1.23 \\ -0.65 & -1.15 & -0.65 & -0.27 & -0.27 & 0.23 & 0.68 & 0.85 & 1.23 \\ -2.21 & -3.71 & -2.21 & -0.82 & 0.68 & 0.68 & 2.12 & 1.79 & 3.68 \\ -2.04 & -3.54 & -2.04 & -0.65 & 0.85 & 0.85 & 1.79 & 2.46 & 2.35 \\ -3.65 & -6.15 & -3.65 & -1.27 & 1.23 & 1.23 & 3.68 & 2.35 & 6.23 \end{bmatrix}.$$

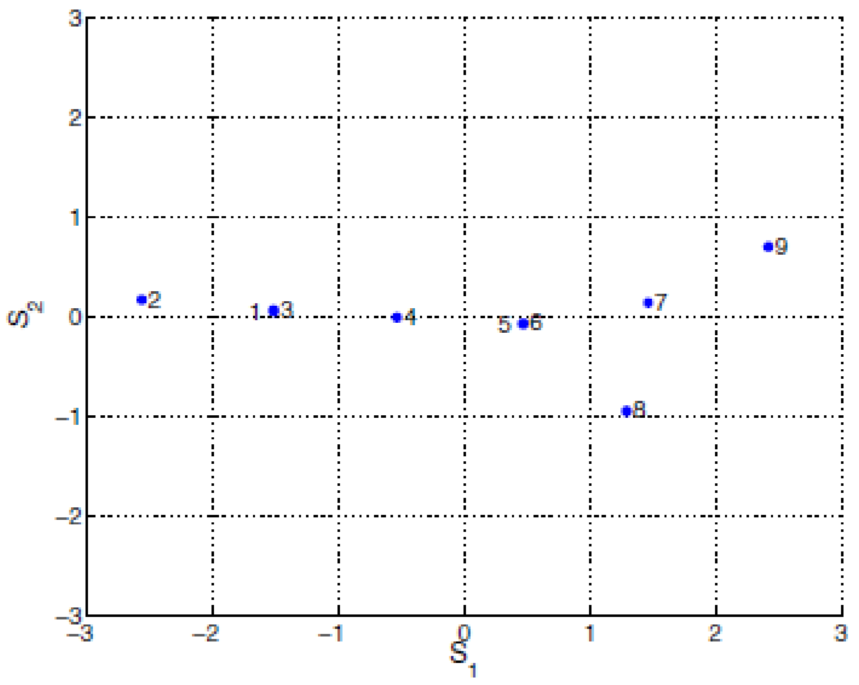
Latent Space models

Suppose we want to map the original network into a 2-dimensional space; we obtain V , Λ, and S:

$$V = \begin{bmatrix} -0.33 & 0.05 \\ -0.55 & 0.14 \\ -0.33 & 0.05 \\ -0.11 & -0.01 \\ 0.10 & -0.06 \\ 0.10 & -0.06 \\ 0.32 & 0.11 \\ 0.28 & -0.79 \\ 0.52 & 0.58 \end{bmatrix}, \quad \Lambda = \begin{bmatrix} 21.56 & 0 \\ 0 & 1.46 \end{bmatrix}, \quad S = V\Lambda^{1/2} = \begin{bmatrix} -1.51 & 0.06 \\ -2.56 & 0.17 \\ -1.51 & 0.06 \\ -0.53 & -0.01 \\ 0.47 & -0.08 \\ 0.47 & -0.08 \\ 1.47 & 0.14 \\ 1.29 & -0.95 \\ 2.42 & 0.70 \end{bmatrix}.$$

The network can be visualized in the 2-dimensional space

Latent Space models



Because nodes 1 and 3 are structurally equivalent, they are mapped into the same position in the latent space.

So are nodes 5 and 6.

k-means can be applied to *S* in order to obtain disjoint partitions of the network. At the end, we obtain two clusters {1, 2, 3, 4}, {5, 6, 7, 8, 9},

$$H = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{bmatrix}.$$

Modularity optimization

- Modularity (Newman, 2006) measures the strength of a community partition for real-world networks by taking into account the degree distribution

- For network of n nodes and m edges, the expected number of edges between nodes v_i and v_j is $d_i d_j/2m$, where d_i and d_j are the degrees of node v_i and v_j , respectively.

  Considering one edge from node v_i connecting to all nodes in the network randomly, it lands at node v_j with probability $d_j/2m$.

  Since v_i has d_i edges, the expected number of connections between the two are $d_i d_j /2m$

  For instance our toy example has 9 nodes and 14 edges. The expected number of edges between nodes 1 and 2 is $3 \times 2/(2 \times 14) = 3/14$.

Modularity optimization

- So $A_{ij} - d_i d_j/2m$ measures how far the observed network interaction between nodes i and j (A_{ij}) deviates from the expected random connections.
- Given a group of nodes C, the strength of community effect is defined as

$$\sum_{i \in C, j \in C} A_{ij} - d_i d_j/2m.$$

- If network is partitioned into k groups the overall community effect can be summed up as:

$$\sum_{\ell=1}^{k} \sum_{i \in C_\ell, j \in C_\ell} \left(A_{ij} - d_i d_j/2m\right).$$

Modularity optimization

- Modularity can be defined as:

$$Q = \frac{1}{2m} \sum_{\ell=1}^{k} \sum_{i \in C_\ell, j \in C_\ell} \left(A_{ij} - d_i d_j / 2m \right).$$

Where 1/2m is introduced to normalize the values between -1/2 and 1 for unweighted and undirected graphs

It is positive if the number of edges within groups exceeds the number expected on the basis of chance. For a given division of the network's vertices into some modules, modularity reflects the concentration of edges within modules compared with random distribution of links between all nodes regardless of modules

- Modularity calibrates the quality of community partitions thus can be used as an objective measure to maximize.

Louvain Method (fast unfolding)

“The Louvain method” is a Heuristic method for greedy modularity optimization and allows to obtain a network community structure with high modularity

The Louvain algorithm is a hierarchical clustering algorithm, that recursively merges communities into a single node and executes the modularity clustering on the condensed graphs. [Blondel et al. Fast unfolding of communities in large networks. J. Stat. Mech. P10008 (2008)]

Algorithm

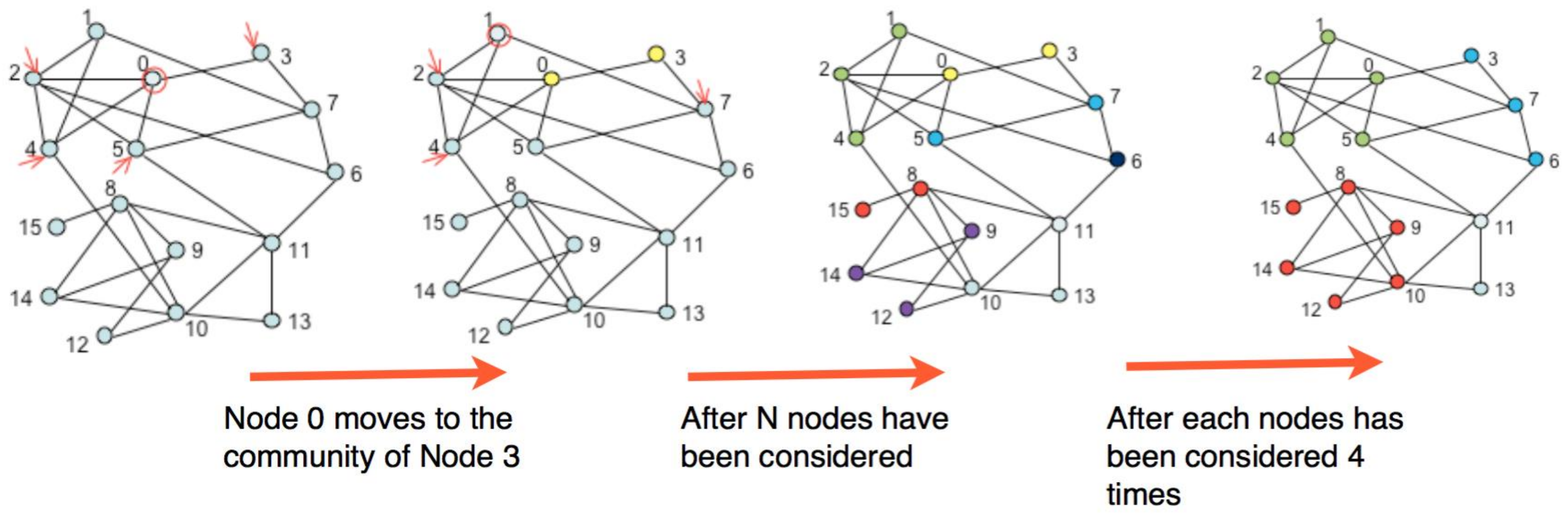
- Assign every node to its own community
- Stage 1: Community Reassignments
  - For every node evaluate modularity gain from removing node from its community and placing it in the community of its neighbor
  - Place node in the community maximizing modularity gain
  - repeat until no more improvement (local max of modularity)
- Stage 2: Coarse Graining
  - Nodes from communities merged into "super nodes“
  - Weight on the links added up
- Repeat until no more changes (max modularity)

Louvain Method (fast unfolding)

"The Louvain method" is agglomerative

The algorithm is based on two steps that are repeated iteratively.

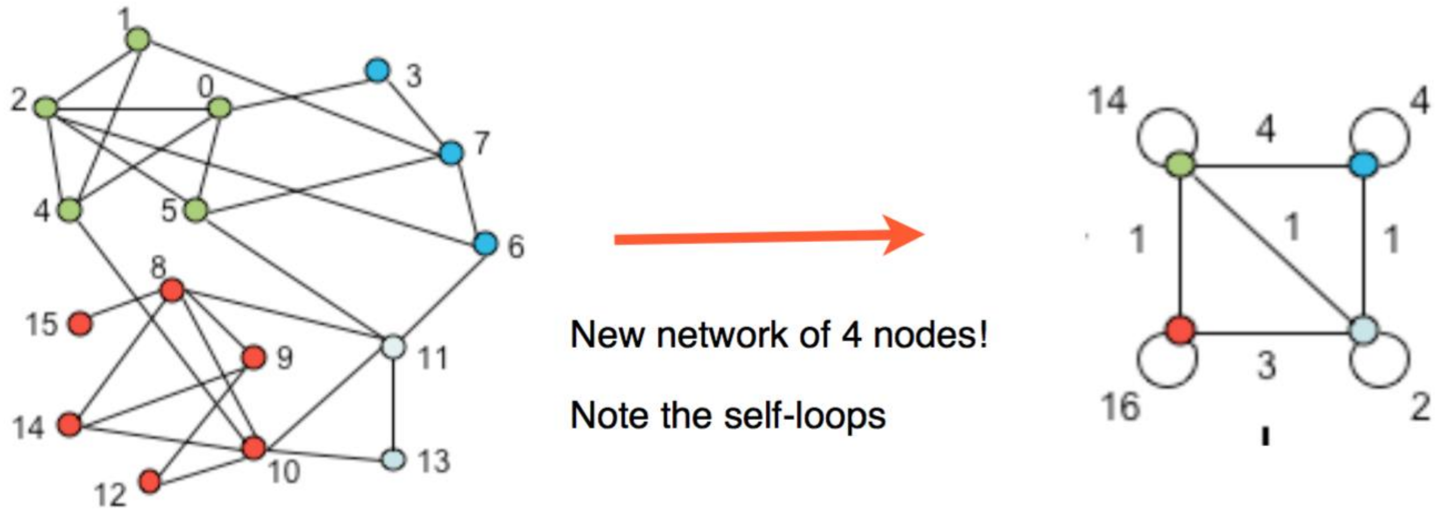
First phase:
Find a local maximum
1) Give an order to the nodes (0,1,2,3,...., N-1)
2) Initially, each node belongs to its own community (N nodes and N communities)
3) One looks through all the nodes (from 0 to N-1) in an ordered way. The selected node looks among its neighbours and adopt the community of the neighbour for which the increase of modularity is maximum (and positive).
4)This step is performed iteratively until a local maximum of modularity is reached (each node may be considered several times).


Node 0 moves to the community of Node 3
After N nodes have been considered
After each nodes has been considered 4 times

Louvain Method (fast unfolding)

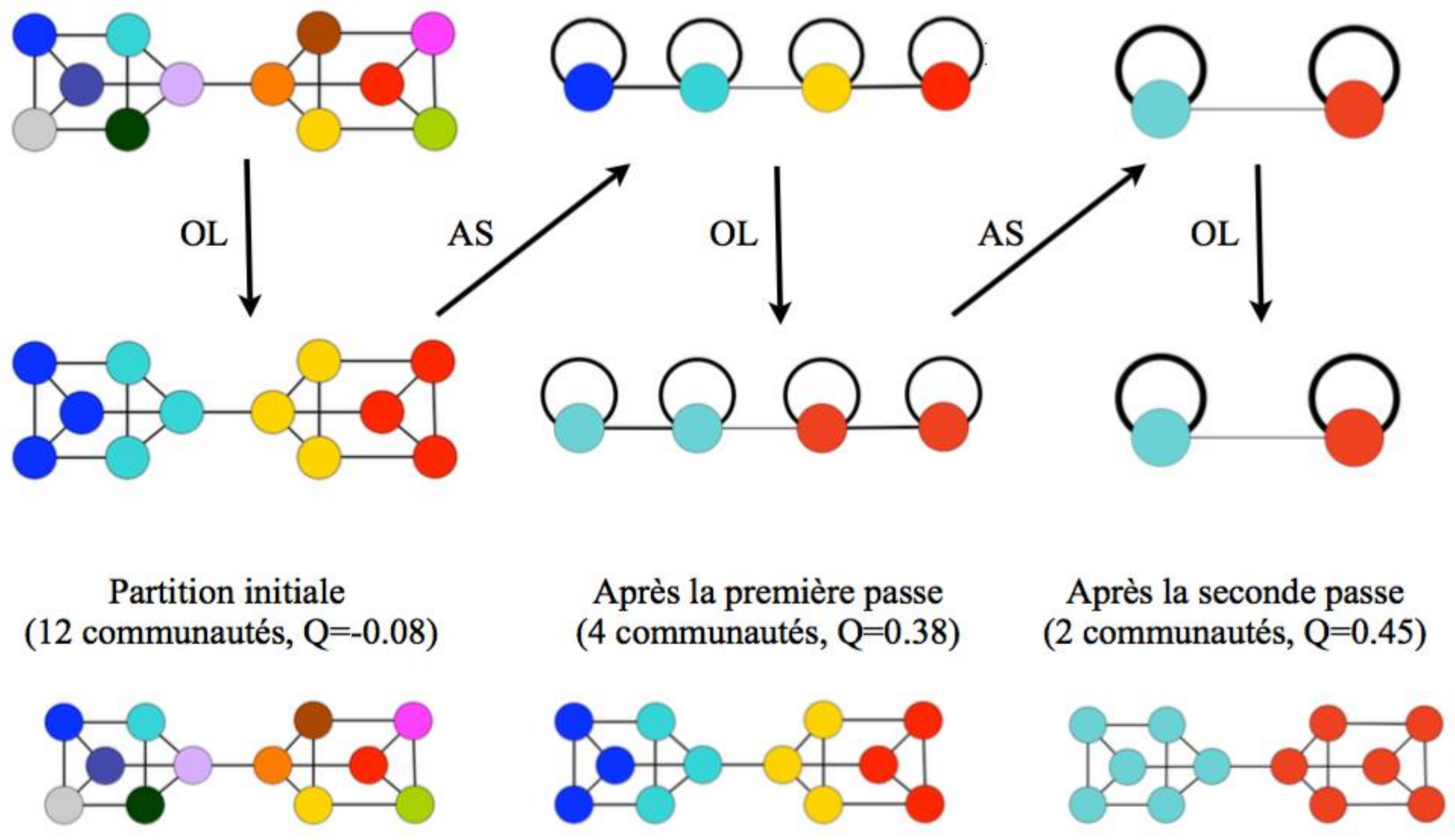
Once a local maximum has been attained, second phase:

We build a new network whose nodes are the communities. The weight of the links between communities is the total weight of the links between the nodes of these communities.



In typical realizations, the number of nodes diminishes drastically at this step.

The two steps are repeated iteratively, thereby leading to a hierarchical decomposition of the network. The phase 1 consists in a local optimization (OL), where each vertex can join one of its direct neighbors community. The second phase consists in a merging of the vertices (AS)
Multi-scale optimisation: local search first among neighbours, then among neighbouring communities, etc.



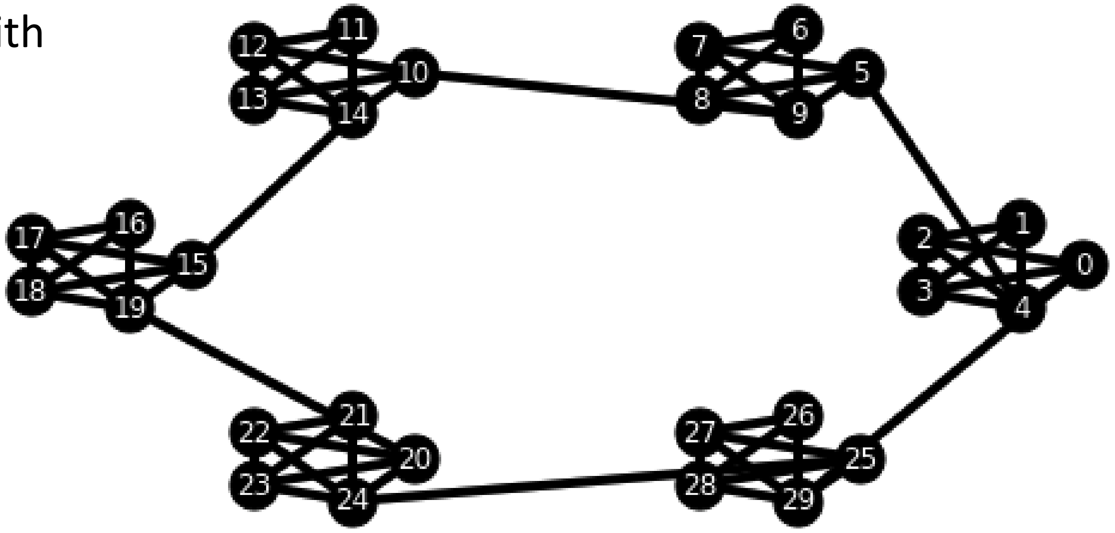
Algorithm 1 Pseudo-code of the community detection algorithm.

```
1: Community detection$G$ initial graph
2: repeat
3:     Place each vertex of $G$ into a single community
4:     Save the modularity of this decomposition
5:     while there are moved vertices do
6:         for all vertex $n$ of $G$ do
7:             $c \leftarrow$ neighboring community maximizing the modularity increase
8:             if $c$ results in a strictly positive increase then
9:                 move $n$ from its community to $c$
10:            end if
11:        end for
12:    end while
13:    if the modularity reached is higher than the initial modularity, then
14:        $end \leftarrow false$
15:        Display the partition found
16:        Transform $G$ into the graph between communities
17:    else
18:        $end \leftarrow true$
19:    end if
20: until $end$
```

Louvain Method example

Let apply Louvain Method to a "connected caveman” graph

This is a network where you begin with N_{cl} fully-connected cliques of M nodes each. Next, you arrange these cliques in a circle. Then, you take one random link from each clique and rewire it so that the clique is connected to its nearest clockwise neighbor. You do this once for each clique, and you end up with something that looks like this:



Louvain Method example

All links in this initial network have unit weight. This is the test network with which we'll explore the Louvain Method below. The "intuitive partition" here consists of the six communities of five nodes each that we've put in by hand.

At the beginning of the Louvain Method, we assign each node to its own community, so in the connected caveman network, there are 30 initial communities, each containing one node.

Louvain Method example

Stage 1: Community Reassignments

In the first stage, we iterate through each of the nodes in the network.

For each node, we consider the change in modularity if we remove the node from its current community and place it in the community of one of its neighbors.

We compute the modularity change for each of the node's neighbors.

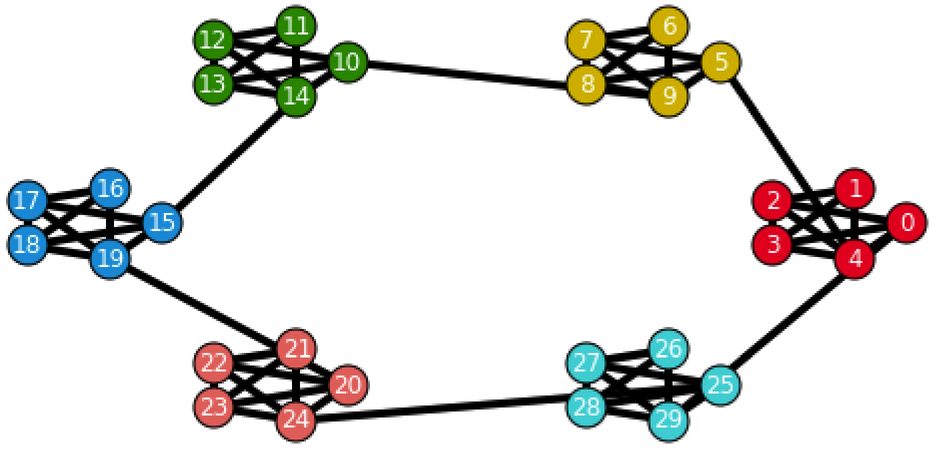
If none of these modularity changes are positive, <u>we keep the node in its current community</u>.

If some of the modularity changes *are* positive, <u>we move the node into the community for which the modularity change is most positive</u>.

We repeat this process for each node until one pass through all nodes yields no community assignment changes.

Louvain Method

Stage 1: Community Reassignments



Louvain Method

Stage 2: Coarse Graining

The next stage in the Louvain Method is to use the communities that were discovered in the community reassignment stage to define a new, coarse-grained network.

In this network, the newly discovered communities are the nodes. The edge weight between the nodes representing two communities is just the sum of the edge weights between the constituent, lower-level nodes of each community.

The links within each community generate self-loops in the new, coarse-grained network.

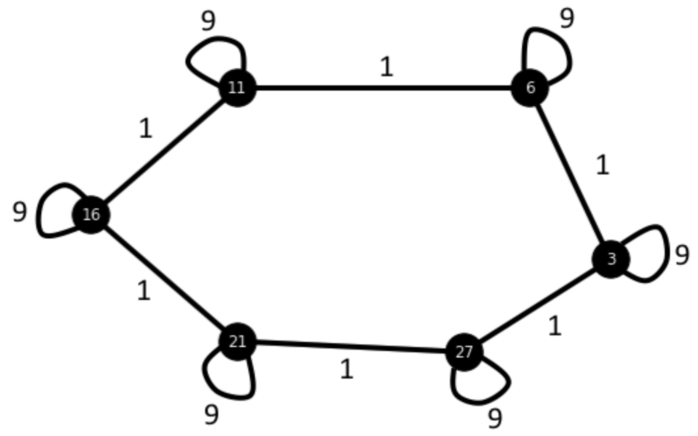
Louvain Method

Stage 2: Coarse Graining

In the simple connected caveman network, there's only one, unit-weight link connecting neighboring communities, so the links between the coarse-grained communities also have unit weight.

If there were two or more links between communities, the new coarse-grained link would have weight equal to the sum of all the lower-level links

Meanwhile, within each community, there are (5×4/2)−1=9 unit-weight links, so the self-loops have weight 9. Here's what the coarse-grained network will look like:



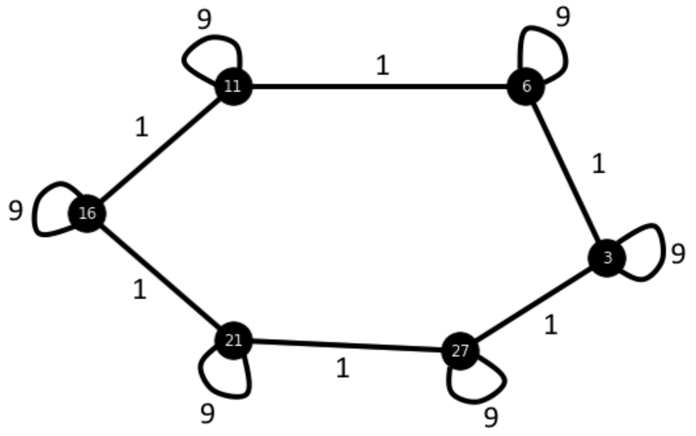
Louvain Method

Stage 3: Repeated Iteration of Stages 1 and 2

The rest of the Louvain Method consists of repeated application of stages 1 and 2. By applying stage 1 (the community reassignment phase) to the coarse-grained graph find a second tier of communities of communities of nodes.

Then, in the next application of stage 2, it defines a new coarse-grained graph at this higher-level of the hierarchy. You keep going like this until an application of stage 1 yields no reassignments. At that point, repeated application of stages 1 and 2 will never yield any more modularity-optimizing changes, so the process is complete.

For the connected caveman graph, the process terminates on the second community reassignment stage.
if we propose a merger of two communities in the coarse-grained graph above. This would result in a negative modularity change.



Louvain Method
Real world application

One of the applications reported in the original Louvain Method paper was a study of a large Belgian phone call network in which nodes represented customers and weighted links represented the number of phone calls between two customers over a six-month period.

The network had 2.6 million nodes and 6.3 million links. The Louvain Method revealed a hierarchy of six levels of communities. At the top level of this hierarchy, the communities representing more than 10,000 customers were strongly segregated by primary language.

All except one of these communities had an 85% or greater majority of either French or Dutch speakers. The sole community with a more equitable distribution was positioned at the interface between French and Dutch clusters in the top-level coarse-grained network. Here's what the authors had to say about this community

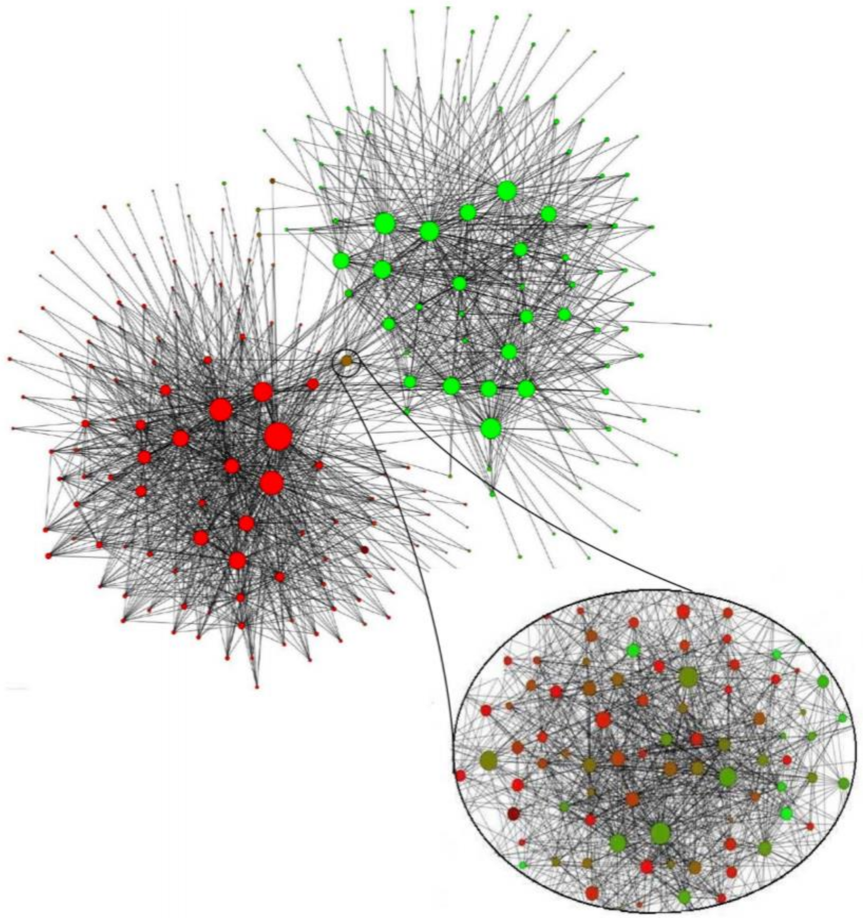


Figure 3. Graphical representation of the network of communities extracted from a Belgian mobile phone network. About 2M customers are represented on this network. The size of a node is proportional to the number of individuals in the corresponding community and its colour on a red-green scale represents the main language spoken in the community (red for French and green for Dutch). Only the communities composed of more than 100 customers have been plotted. Notice the intermediate community of mixed colours between the two main language clusters. A zoom at higher resolution reveals that it is made of several sub-communities with less apparent language separation.

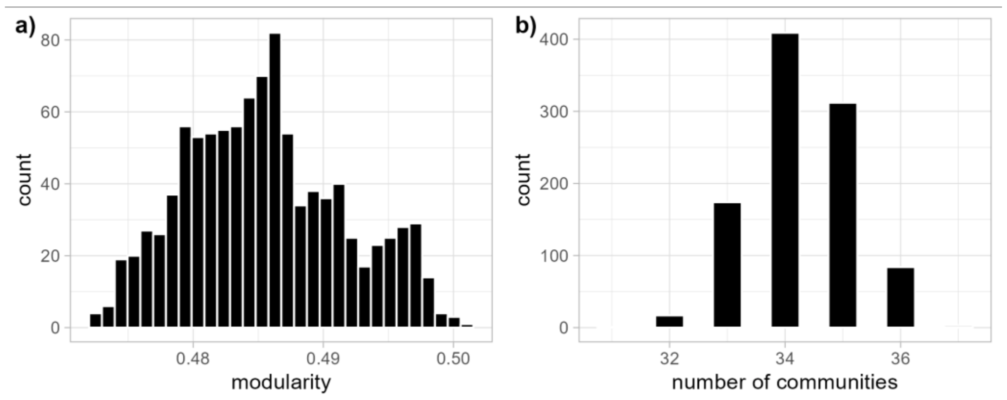
Issues in modularity maximization

The Louvain Method, and modularity optimization algorithms more generally, have found wide application across many domains. However, fundamental problems with these algorithms have also been identified

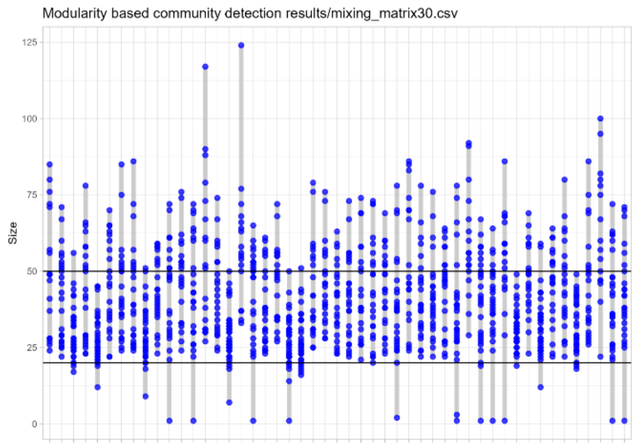
1) **The "resolution" limit:** When pass through the community modification and coarse-graining stages several of the intuitive communities are merged together. This is a general problem with modularity optimization algorithms. They have trouble detecting small communities in large networks.
2) **The "degeneracy" problem**: There are typically an exponentially large (in network size) number of community assignments with modularities close to the maximum. This can be a severe problem because, in the presence of a large number of high modularity solutions, it's (a) hard to find the global maximum and (b) difficult to determine if the global maximum is truly more scientifically important than local maxima that achieve similar modularity.

Issues in modularity based community detection

At each trial of the modularity-based community detection, we obtain a different set of communities (and corresponding modularity)



Distribution of modularity and number of communities in the labour-market network



Community size over 100 independent trials in a LFR reference network.

Network centric CD – Modal clustering for networks

Method proposed to detect **complex shaped communities** (e.g. leadership based communities)
[Menardi & De Stefano Density-based clustering of social networks JRSS A (2022)]

Idea: different measures of actor centrality account for different node roles

Specificity of modal clusters for network data, while accounting for:

- different strength of relationships

  weighted networks

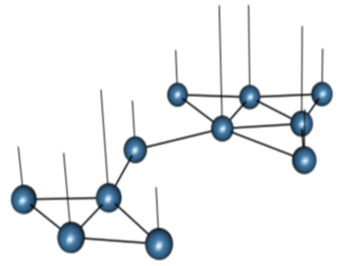
- different kinds of aggregation mechanisms

  different measures of centrality of actors (used as density function)
  though any function capturing the connectivity of nodes can be used

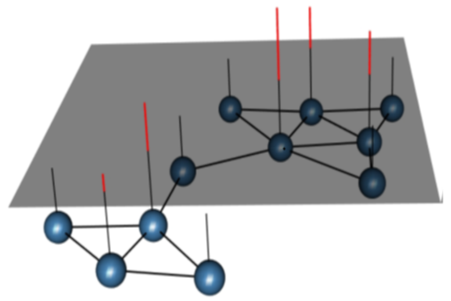
Network centric CD – Modal clustering for networks

Rationale.

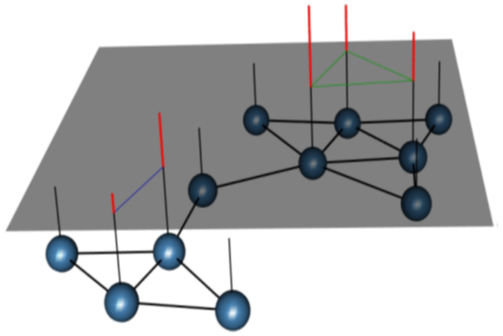
1) compute density on the nodeset



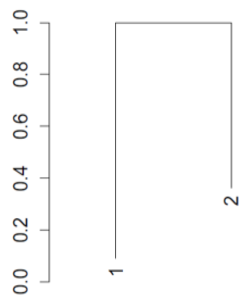
2) identify the level sets



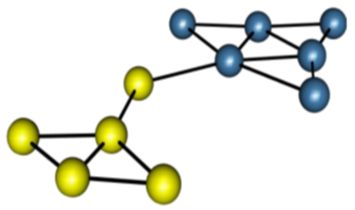
3) Communities are the connected components of the subnetwork induced by the level sets



4) by varying the level at which the contour set is evaluated the cluster tree is created



the number of communities is the number of leaves of the cluster tree



Hierarchy-centric

- Another line of community detection research is to build a hierarchical structure of communities based on network topology.

- This facilitates the examination of communities at different granularity.

- There are mainly two types of hierarchical clustering: divisive, and agglomerative.

- One particular divisive clustering algorithm receiving much attention is to recursively remove the “weakest” tie in a network until the network is separated into two or more components.

Hierarchy-centric

- Newman and Girvan (2004) proposes to find the weak ties based on edge betweenness.
- Edge betweenness is highly related to the betweenness centrality
- It is defined to be the number of shortest paths that pass along one edge

If two communities are joined by only a few cross-group edges, then all paths through the network from nodes in one community to the other community have to pass along one of these edges.

- The number of shortest paths is expected to be large for between-group edges.

Girvan-Newman algorithm

- The Girvan–Newman algorithm detects communities by progressively removing edges from the original network. The connected components of the remaining network are the communities (Newman and Girvan: Finding and evaluating community structure in networks, *Physical Review E* 69, 026113, 2004)
- the Girvan–Newman algorithm focuses on edges that are most likely "between" communities.
- The idea was to find which edges in a network occur most frequently between other pairs of nodes by finding edge betweenness.
- it is likely that edges connecting separate communities have high edge betweenness as all the geodesic from one group to another must pass through them.
- So if we gradually remove the edge with the highest edge betweenness score we will get a hierarchical map (dendrogram) of the graph. The leafs of the tree are the individual vertices and the root of the tree represents the whole graph.

Girvan-Newman algorithm

We can express Girvan-Newman algorithm in the following procedure:

1) Calculate edge betweenness for every edge in the graph.

2) Remove the edge with highest edge betweenness.

3) Calculate edge betweenness for remaining edges.

4) Repeat steps 2–4 until all edges are removed (or we obtain connected components at certain level)

Girvan-Newman algorithm

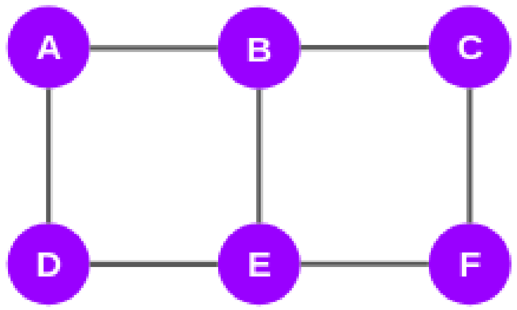
- In order to calculate edge betweenness it is necessary to find all shortest paths (geodesic) in the graph. The algorithm starts with one vertex, calculates edge weights for paths going through that vertex
- then repeats it for every vertex in the graph and sums the weights for every edge.

- The edge betweenness centrality (EBC) can be defined as the number of shortest paths that pass through an edge in a network. Each and every edge is given an EBC score based on the shortest paths among all the nodes in the graph.

- We can compute a score for each edge based on this concept by traversing the network from each of the ego perspective and traversing back the graph

Girvan-Newman example

- Let’s take an example to find how EBC scores are calculated. Consider this graph below with 6 vertices and 7 edges



Girvan-Newman example

First step: Calculate the edge betweenness

find the EBC scores for all the edges is an iterative process and it works taking one node at a time and plot the shortest paths to the other nodes from the selected node

Based on the shortest paths, then compute the EBC scores for all the edges

repeat this process for every node in the graph.

Therefore, there will be 6 iterations of this process

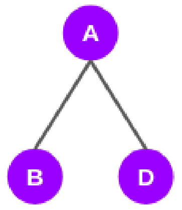
This means every edge will get 6 scores. These scores will be added edge-wise

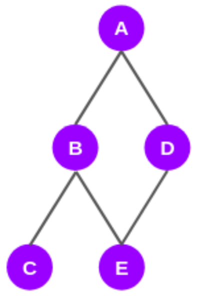
Finally, the total score of each edge will be divided by 2 to get the EBC score

Girvan-Newman example

Iteration 1 (starting from node A)

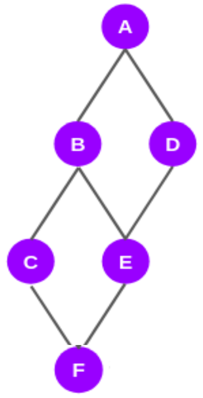
A is directly connected nodes to node A are nodes B and D. So, the shortest paths to B and D from A are AB and AD respectively





The shortest paths to nodes C and E from A go through B and D

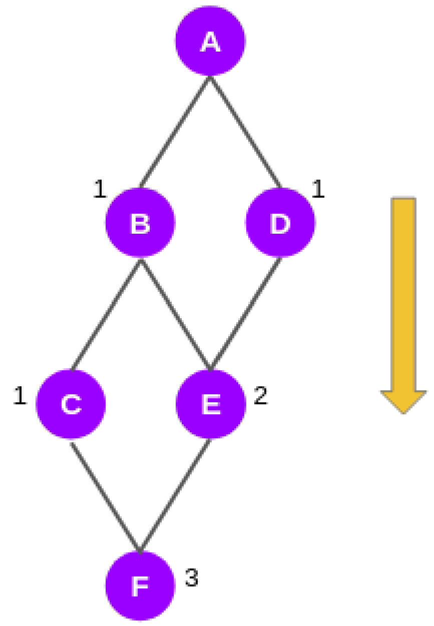
The shortest paths to the last node F from node A, pass through nodes B, D, C, and E



The graph above depicts only the shortest paths from node A to all the other nodes.

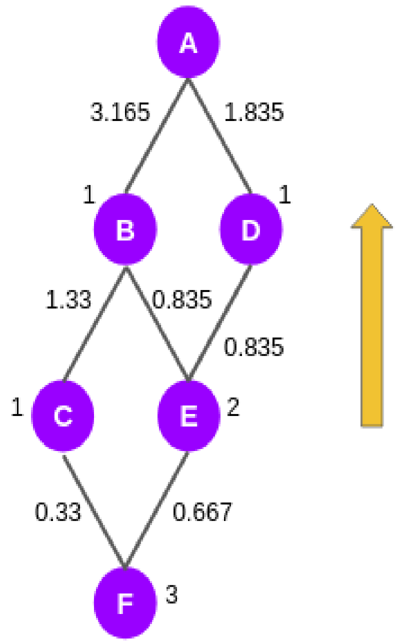
Girvan-Newman example

Before giving scores to the edges, we will assign a score to the nodes in the shortest-path-graph. To assign these scores, we will have to traverse the graph from the root node, i.e., node A to the last node (node F)



Girvan-Newman example

Computing Scores for Edges moving in the backward direction, from node F to node A



FC = ⅓ = 0.33
FE = ⅔ = 0.667

CB = 1 + 0.33 = 1.33
EB = (1 + 0.667)/2 = 0.835
ED = (1 + 0.667)/2 = 0.835

BA = (1 + 1.33 + 0.835)/1 = 3.165
DA = (1 + 0.835)/1 = 1.835

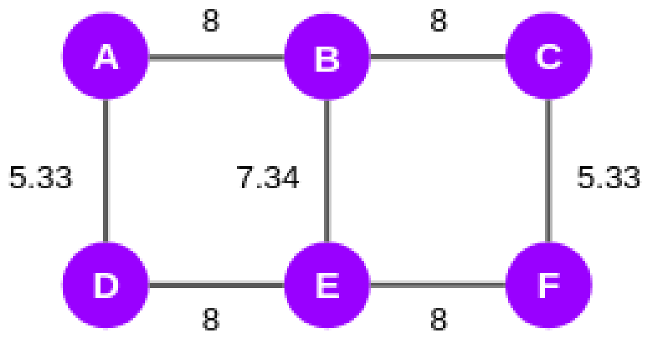
1) We first compute the score for the edges FC and FE.

The edge score for edge FC is the ratio of the node scores of C and F, i.e. 1/3 or 0.33. Similarly, for FE the edge score is 2/3.

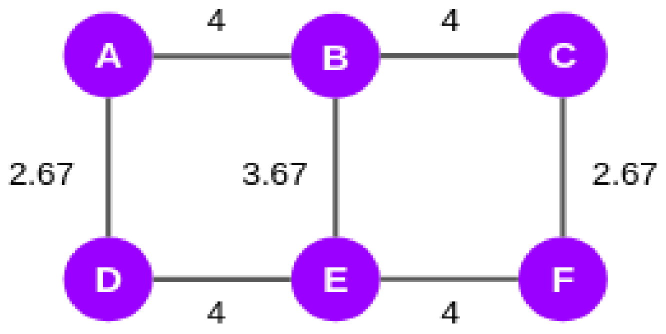
2) calculate the edge score for the edges CB, EB, and ED.
from this level onwards, every node will have a default value of 1 and the edge scores computed in the previous step will be added to this value.
So, the edge score of CB is (1 + 0.33)/1. Similarly, edge score EB or ED is (1 + 0.667)/2. Then we move to the next level to calculate the edge scores for BA and DA.

Girvan-Newman example

We will have to repeat the same steps again from the other remaining five nodes. In the end, we will get a set of six scores for all the edges in the network.



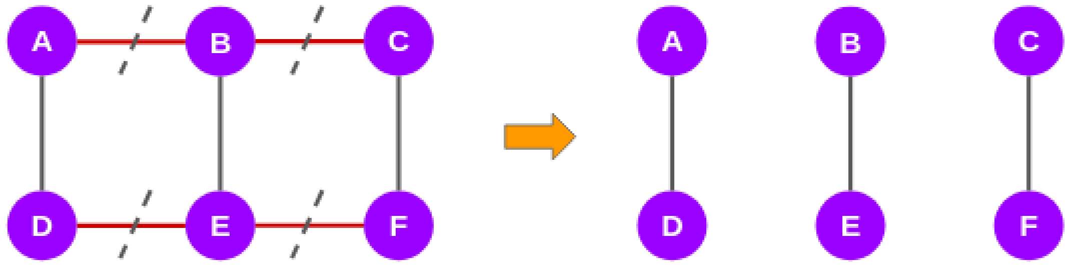
Since it is an undirected graph, we will divide these scores by two and finally, we will get the EBC scores:



Girvan-Newman example

According to the Girvan-Newman algorithm, after computing the EBC scores, the edges with the highest scores will be taken off till the point the graph ideally splits into different components

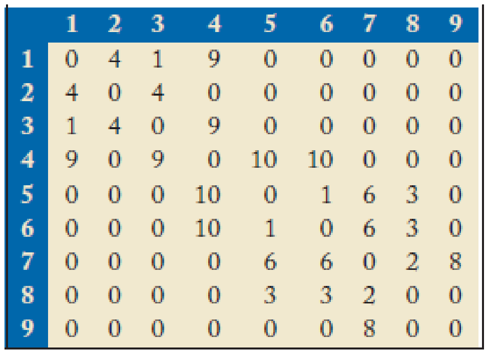
we can see that the edges AB, BC, DE, and EF have the highest score, i.e., 4. We will strike off these edges and it gives us 3 subgraphs that we can call communities



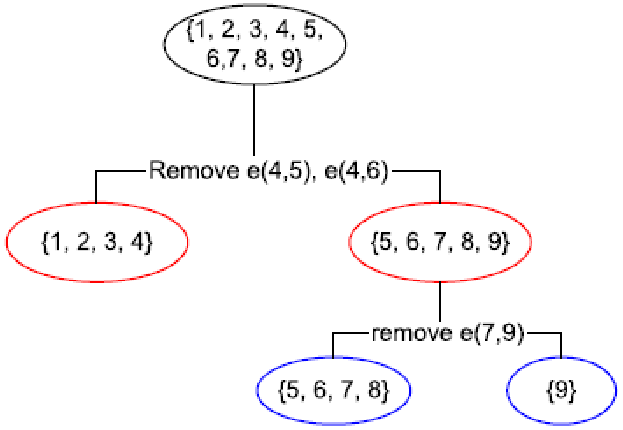
Girvan-Newman (toy example)

You can see GN algorithm as a hierarchical clustering algorithm.

You compute the EBC after each disaggregation



	1	2	3	4	5	6	7	8	9
1	0	4	1	9	0	0	0	0	0
2	4	0	4	0	0	0	0	0	0
3	1	4	0	9	0	0	0	0	0
4	9	0	9	0	10	10	0	0	0
5	0	0	0	10	0	1	6	3	0
6	0	0	0	10	1	0	6	3	0
7	0	0	0	0	6	6	0	2	8
8	0	0	0	0	3	3	2	0	0
9	0	0	0	0	0	0	8	0	0



Overlapping communities

- There are several C.D. methods for the identification of overlapping communities
- The motivaton is that in social networks individual actors may participate to diverse closed and dense groups
- One community detection algorithm that is aimed at identifying overlapping communities is the clique percolation algorithm, which has been developed for unweighted undirected networks by Palla, Derényi, Farkas, & Vicsek, 2005

Clique percolation

The clique percolation algorithm for unweighted networks proceeds as follows:

1) Find out all cliques of size k in the given network;

2) Construct a clique graph. Two cliques are adjacent if they share $k - 1$ nodes;

3) Each connected component in the clique graph is a community.

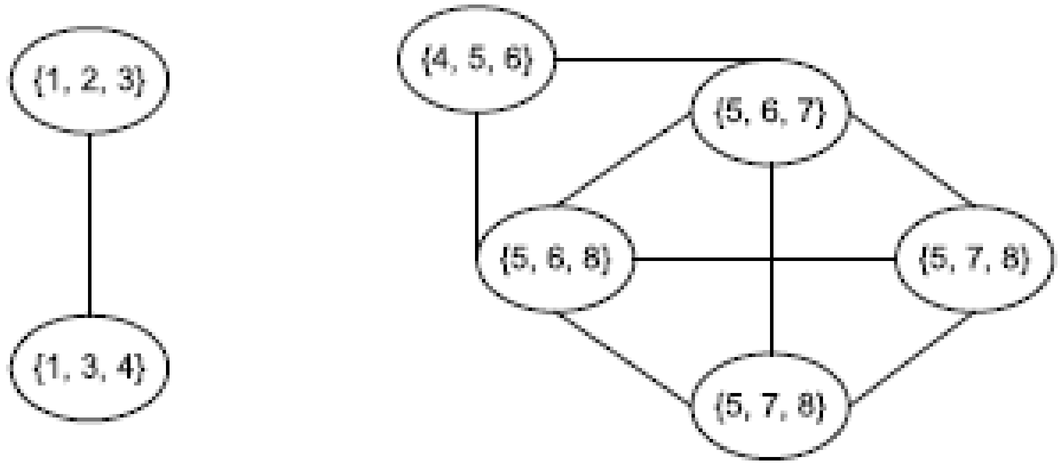
Clique percolation

Take the network in the toy example.

For k = 3, we can identify all the cliques of size 3 as follows.

{1, 2, 3}	{1, 3, 4}	{4, 5, 6}	{5, 6, 7}	{5, 6, 8}	{5, 7, 8}	{6, 7, 8}

Then we have the clique graph where 2 cliques are connected if they share k – 1 (2 in our case) nodes.



Clique percolation

In the clique graph, there are two connected components.

The nodes in each component fall into one community.

Consequently, we obtain two communities: {1, 2, 3, 4} and {4, 5, 6, 7, 8}.

Note that node 4 belongs to both communities. In other words, we obtain two overlapping communities.

Note that the clique percolation method requires the enumeration of all the possible cliques of a fixed size k.

This can be computational costly for large-scale networks.

Community evaluation

The reason for so many assorted definitions and methods, is that there is no clear ground truth information about a community structure in a real world network.

Therefore, different community detection methods are developed from various applications of specific needs.

Strategies commonly adopted to evaluate identified communities in order to facilitate the comparison of different community detection methods are:

1) Groups with self-consistent *structural definitions*.
Some groups like cliques, k-cliques, k-clubs, and k-cores can be examined immediately once a community is identified. We can simply check whether the extracted communities satisfy the definition.

Community evaluation

2) Networks with ground truth. That is, the community membership for each actor is known. Ideal scenario (ex: Karate club network).

In presence of ground truth a way to evaluate community detection results is to consider all the possible pairs of nodes and check if they belong to the same community.

It is considered an error if two nodes of the same community are assigned to different communities, or two nodes of different communities are assigned to the same community.

Let $C(v_i)$ denote the community of node v_i. We can construct a contingency table: a, b, c and d are frequencies of each case.

Community evaluation

		Ground Truth	
		$C(v_i) = C(v_j)$	$C(v_i) \neq C(v_j)$
Clustering Result	$C(v_i) = C(v_j)$	a	b
	$C(v_i) \neq C(v_j)$	c	d

For ex.: a is the frequency that two nodes are assigned into the same community in the ground truth as well in the CD result.

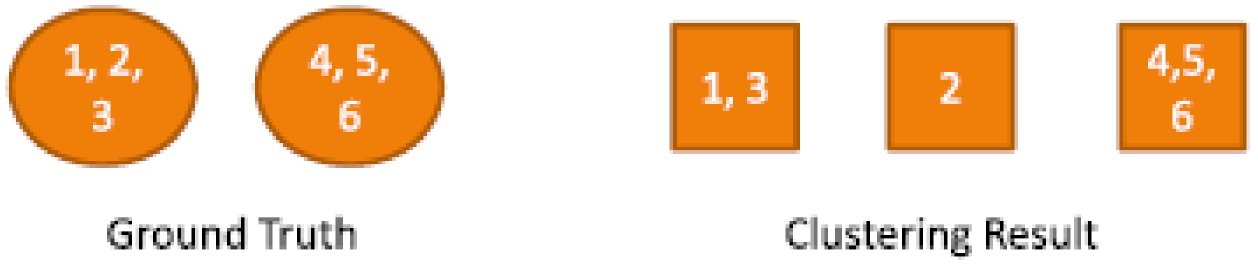
Note that the total sum of frequencies is the number of all possible pairs of nodes in the network: $a + b + c + d = n(n - 1)/2$.

Based on the frequencies, the accuracy of CD algorithm can be computed as

$$accuracy = \frac{a+d}{a+b+c+d} = \frac{a+d}{n(n-1)/2}.$$

Community evaluation

Example result



We have a = 4. Specifically, {1, 3},{4, 5}, {4, 6}, {5, 6} are assigned into the same community in the ground truth and CD result.

Any pair between {1, 2, 3} and {4, 5, 6} are being assigned to different communities, thus d = 9.

Consequently, the accuracy of the clustering result is

$$(4 + 9)/(6 \times 5/2) = 13/15$$

Community evaluation

3) Networks without ground truth or other information.

This is the most common situation, yet it requires objective evaluation most. Normally, one resorts to some quantitative measure for network validation. For instance modularity

It can be use a similar procedure as cross validation in classification for validation.
It extracts communities from a (training) network and then compares them with those of the same network (e.g., constructed from a different date) or another related network based on a different type of interaction.

In order to quantify the quality of extracted community structure, a common measure being used is modularity.
The method with higher modularity is selected.

Community evaluation

4) Artificial networks benchmark

Existing evaluation tests and benchmarks involves as said Small networks with known community structure.

But we can also simulate Artificial graphs with simplified structure.

Planted l-partition model

- Partition the graph with **N** nodes into **N/l** Partitions.
- Each node has a probability $\mathbf{p}_{in}$ of being connected to nodes of its group
- and a probability $\mathbf{p}_{out}$ of being connected to nodes of different groups.
- As long as $\mathbf{p}_{in} \geq \mathbf{p}_{out}$ the graph has a community structure else it's a Random Graph.

Community evaluation

LFR Benchmark

- A special case of Planted l-partition model, in which groups have different size and nodes have different degrees.
- Node degree distribution based on power law with exponent $\boldsymbol{\tau_1}$
- Community size also obeys power law distribution with exponent $\boldsymbol{\tau_2}$
- Each node receives its degree which remains the same throughout.
- Mixing parameter μ, is the ratio of external degree of a node with respect to its community and the total degree of the node.
- For simplicity all nodes have the same μ.

Community evaluation

LFR Benchmark

Based on power law distribution with exponent $\boldsymbol{\tau_2}$ the sizes of the communities are assigned (Sum matches the size **N** of the network).

Each community is treated as an isolated graph.

Assign degree $\mathbf{k}_i$ to a node $\boldsymbol{i}$ based on power law distribution with exponent $\boldsymbol{\tau_1}$.

Assign internal degree $\mathbf{(1- \mu)\ k}_i$ to node $\boldsymbol{i}$.