Community Detection

Algorithm Design

Zhemin Huang, Xunjie Wang

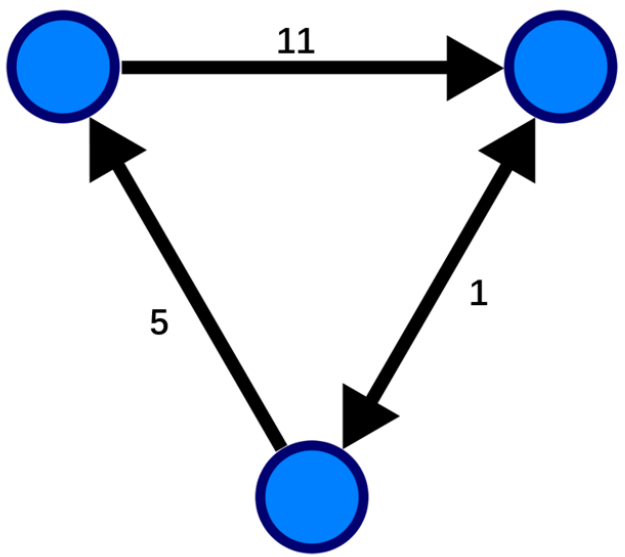
School of Software, Shanghai Jiao Tong University

Problem Description

Network

Definition (Network)

A network is represented as $G = (V, E, W)$,
where V, E denote the set of nodes and edges, respectively, and W denotes the corresponding weights of the connections.



Community

Definition (Community)

Communities are the sub-graphs in a network, where nodes share dense connections. Sparsely-connected nodes cripple communities.

We use $C = \{C_1, C_2, ..., C_k\}$ to denote a set of k communities divided from a network G. A node v clustered into the community C_i satisfies the condition that the internal degree of each node inside the community exceeds its external degree.

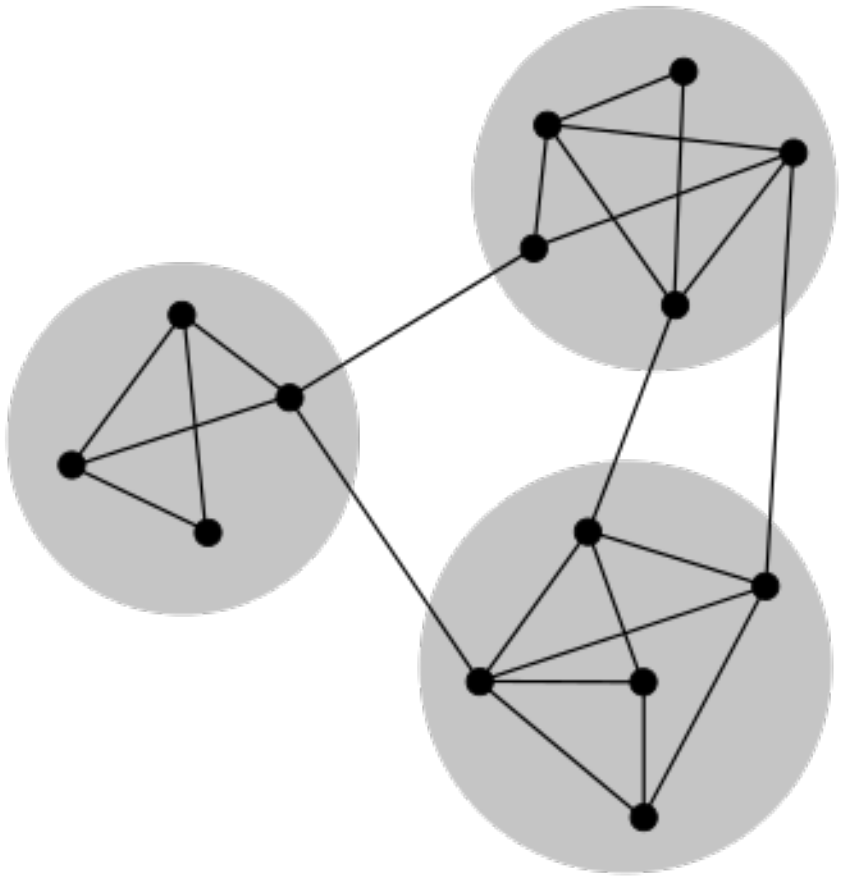
For $\forall i, j, C_i \in C, C_j \in C$, they satisfy the following conditions

$$C_i \cap C_j = \varnothing$$

$$\bigcup_{i=1}^{k} C_i = G$$

Community

A community is a type of sub-graph that represents some real social phenomenon. In order words, a community is a group of people or objects, which share common characteristics.



Modularity

Definition (Modularity)

Consider a particular division of a network into k communities. Let us define a $k \times k$ symmetric matrix e whose element e_{ij} is the fraction of all edges in the network that link vertices in community i to vertices in community j.

We further define the row sums $a_i = \sum_j e_{ij}$, which represent the fraction of edges that connect to vertices in community i.

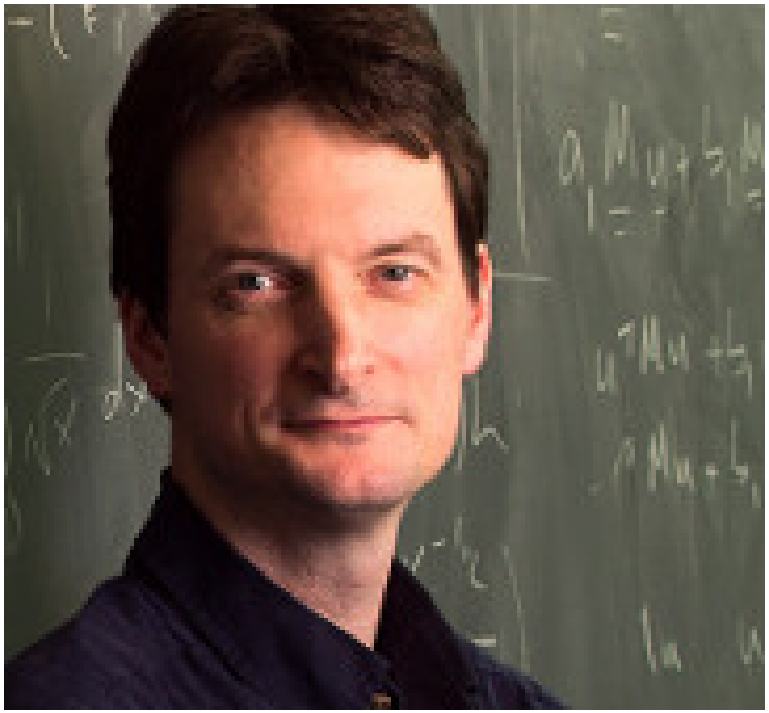
In a network in which edges fall between vertices who belong to different communities, we would have $e_{ij} = a_i a_j$. Let $||X||$ indicates the sum of the elements of the matrix X. Thus, we define modularity by

$$Q = \sum_i e_{ii} - {a_i}^2 = \sum_i e_{ii} - \sum_i {a_i}^2 = Trace - ||e^2||$$

Modularity

Proposed by Mark Newman, UMich.

He is known for his fundamental contributions to the fields of complex networks and complex systems, for which he was awarded the 2014 Lagrange Prize.

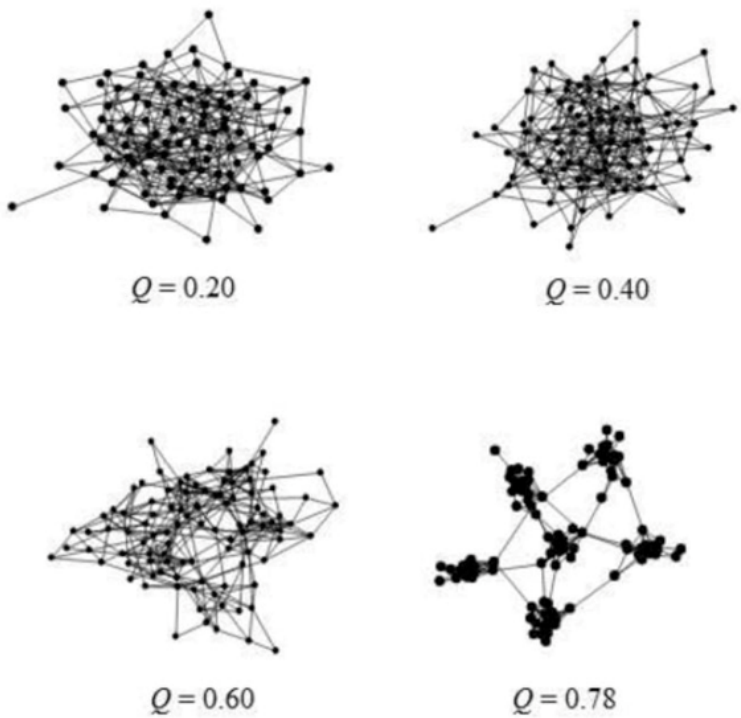


Modularity

Modularity was designed to measure the strength of division of a network into communities.

If Q is high, we get strong community structure.

In practice, values for such networks typically fall in the range from about 0.3 to 0.7.



Reformulation of The Modularity

In order to raise a spectral algorithm for community detection, Newman redefined the concept of modularity.

Definition (Reformulation of The Modularity)

The modularity of a community partition is a scalar ranging from $-\frac{1}{2}$ to 1 that evaluates the density of links inside communities as compared to links between communities.

For a given graph G. Let A_{ij} be the weight of the edge between i and j, $k_i = \sum_{(i,j)\in E} A_{ij}$ is the sum of the weights of all the edges attached to vertex i, $m = \frac{1}{2}\sum_{ij} A_{ij}$, the quantity $\frac{1}{2}(s_i s_j + 1)$ is 1 if i and j are in the same group and 0 otherwise,

$$Q = \frac{1}{4m} \sum_{i\in V, j\in V} (A_{ij} - \frac{k_i k_j}{2m}) \frac{1}{2}(s_i s_j + 1)$$

The leading factor of $\frac{1}{4m}$ seems to be confusing. Actually, it is included for compatibility with the previous definition of modularity.

Community Detection

From the definitions above, we can lead to our goal – community detection.

Definition (Community detection)

Community detection is a method to extract communities from large networks when the modularity is optimal.

Not all methods are based on modularity, such as Infomap.

Modularity has its limits, but it is still the most commonly used method for evaluating results.

Key Properties

Modularity

Lemma

We denote the set of all possible communities of a graph G with $A(G)$, $Q(P)$ refers to the modularity of P.
Let G be an undirected and unweighted graph and $P \in A(G)$.
Then $-\frac{1}{2} \leq Q(P) \leq 1$ holds.

Proof

Proof. Let $m_i = |E(P)|$ be the number of edges inside community P and $m_e = \sum_{P \neq P' \in P} |E(P, P')|$ be the number of edges having exactly one end-node in P.
Then we can calculate the contribution of P to $Q(P)$

$$\frac{m_i}{m} - (\frac{m_i}{m} + \frac{m_e}{2m})^2 = \frac{-4(m_i)^2 + 4m_i(m - m_e) - (m_e)^2}{4m^2}$$

It is obvious that the only maximum point is at $m_i = \frac{m - m_e}{2}$. The contribution of a community is minimized when m_i is zero and m_e is as large as possible. Suppose now $m_i = 0$, the upper bound can only be actually attained in the specific case of a graph with no edges, where coverage is defined to be 1. Besides, any bipartite graph $K_{a,b}$ with simple communities $C = \{C_a, C_b\}$ yields the minimum modularity of $\frac{1}{2}$. This proof the lemma.

Modularity

Theorem

Modularity is **NP-complete**.

Proof(1)

We formalize the problem of finding modularity, and prove it by **reduction**.

Problem 1 (Modularity)

Given a graph G and a number K, is there a community $P \in G$, for which $Q(P) \geq K$? $(K \in [-\frac{1}{2}, 1])$?

Problem 2 (3-Partition)

Given $3k$ positive integer numbers $a_1, ..., a_{3k}$ such that the sum $\sum_{i=1}^{3k} a_i = kb$, and $\frac{b}{4} < a_i < \frac{b}{2}$, for an integer b and for all $i = 1, ..., 3k$, is there a partition of these numbers into k sets, such that the sum of the numbers in each set equals to b?

Proof(2)

It has been proved that 3-partition problem is a NP-Complete problem.

Then we show that an instance $A = a_1, ..., a_{3k}$ of 3-partition problem can be transformed into an instance $G(A), K(A)$ of modularity problem.

Given an instance A of 3-partition, we can construct a graph $G(A)$ with k cliques $H_1, ..., H_k$ of size $a = \sum_{i=1}^{3k} a_i$ each. For each element $a_i \in A$, we introduce a single element node in graph, and connect it to a_i nodes in each of the k cliques, where each clique member is connected to exactly one element node. Therefore, each clique node has degree a, and the element node corresponding to element $a_i \in A$ has degree ka_i. The number of edges in $G(A)$ is $m = \frac{k}{2}a(a+1)$.

Proof(3)

Then we construct $K(A)$. Since graph $G(A)$ has exactly k cliques, it has exactly $(k-1)a$ inter-community edges, so the edge contribution is given by

$$\sum_{C \in P} \frac{|E(C)|}{m} = \frac{m-(k-1)a}{m} = 1 - \frac{2(k-1)a}{ka(a+1)} = 1 - \frac{2k+2}{k(a+1)}$$

Therefore, communities $P = (C_1, ..., C_k)$ with maximum modularity must minimize $d(C_1)^2 + d(C_2)^2 + ... + d(C_k)^2$, where $d(C_k)$ refers to the degree of C_k. Then the sum of degrees per community should be as small as possible.

In the optimum case, we can assign to each community element nodes corresponding to elements that sum to $b = \frac{1}{k}a$. In each clique, the sum equals to $k\frac{1}{k}a = a$.

$$d(C_1)^2 + ... + d(C_k)^2 \geq k(a^2 + a)^2 = ka^2(a+1)^2$$

Proof(4)

Hence, if there exist communities P with $Q(P)$, then

$$K(A) \geq 1 - \frac{2k-2}{k(a+1)} - \frac{ka^2(a+1)^2}{k^2a^2(a+1)^2} = \frac{(k-1)(a-1)}{k(a+1)}$$

As each element node is contained in exactly one community, this yields a solution for the instance of 3-partition. The instance of 3-partition is satisfiable if the instance of modularity is satisfiable.

Otherwise, suppose the instance for 3-partition is satisfiable. Then there exists a partition into k sets, where the sum over each set is $\frac{1}{k}a$. If we detect communities by joining the element nodes of each set with a different clique, we get communities of modularity $K(A)$. Therefore, the instance of modularity is satisfiable if the instance of 3-partition is satisfiable.

Therefore, the theorem holds.

Girvan-Newman Algorithm

Edge Betweenness

Definition

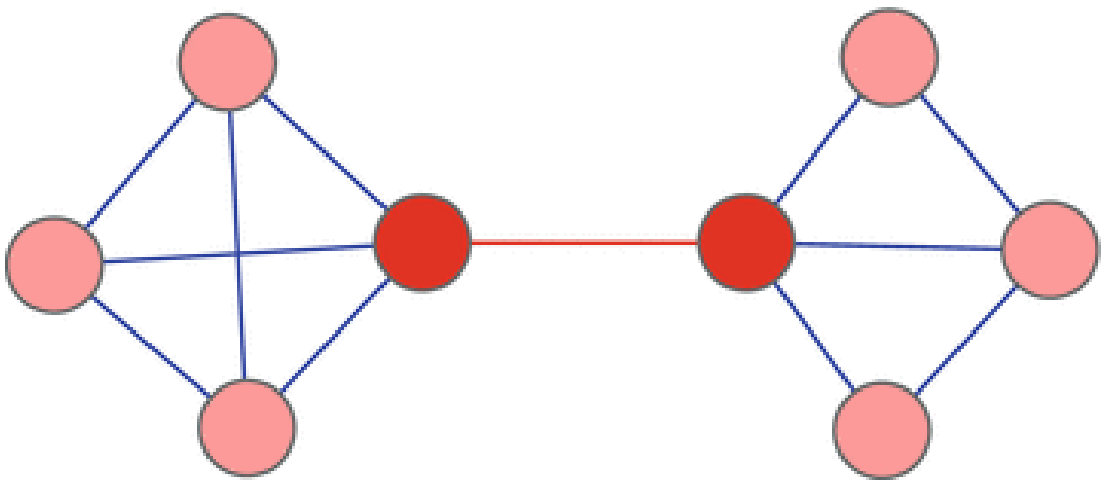
Edge betweenness of an edge (i, j) is the number of the shortest paths between pairs of vertices that pass through the edge (i, j).

Edge Betweenness

Definition

Edge betweenness of an edge (i, j) is the number of the shortest paths between pairs of vertices that pass through the edge (i, j).

Intuition. An edge with a high edge betweenness score represents a bridge-like connector between two parts of a network, and the removal of which may affect the communication between many pairs of nodes through the shortest paths between them.



Edge Betweenness

- $d_{s,i}$ is the length of the shortest path between vertex s and vertex i.
- $w_{s,i}$ is the number of the shortest paths from vertex s to vertex i.
- $b_{s,i}$ is the number of the shortest paths between vertex s to any vertex in graph that pass through vertex i.

Edge Betweenness

- $d_{s,i}$ is the length of the shortest path between vertex s and vertex i.
- $w_{s,i}$ is the number of the shortest paths from vertex s to vertex i.
- $b_{s,i}$ is the number of the shortest paths between vertex s to any vertex in graph that pass through vertex i.

Edge betweenness $\sigma_{i,j}$ of edge (i,j):

$$\sigma_{i,j} = \sum_{s \in V} \sigma_{s,i,j}$$

where

$$\sigma_{s,i,j} = \begin{cases} \frac{w_{s,j}}{w_{s,i}} b_{s,i} & d_{s,i} > d_{s,j} \\ \frac{w_{s,i}}{w_{s,j}} b_{s,j} & d_{s,i} < d_{s,j} \\ 0 & d_{s,i} = d_{s,j} \end{cases}$$

Breath First Search

Breath-First-Search($G = (V, E), s$)

$d_{s,s} \leftarrow 0, w_{s,s} \leftarrow 1, b_{s,s} \leftarrow 0,\ Q \leftarrow \{s\}, L \leftarrow \{s\}$;
for *each vertex* $v \in V - \{s\}$ **do**
    $d_{s,v} \leftarrow \infty, w_{s,v} \leftarrow 0, b_{s,v} \leftarrow 1$;
end
while Q *is not empty* **do**
    Dequeue $i \leftarrow Q$;
    for *each vertex* j *where* $(i, j) \in E$ **do**
        if $d_{s,j} \neq \infty$ *and* $d_{s,j} = d_{s,i} + 1$ **then**
            $w_{s,j} = w_{s,j} + w_{s,i}$;
        end
        if $d_{s,j} = \infty$ **then**
            $d_{s,j} = d_{s,i} + 1, w_{s,j} = w_{s,i}$, Enqueue $j \rightarrow Q$, Push $j \rightarrow L$;
        end
    end
end
Return (d, w, s, L);

Reverse Breath First Search

Reverse-Breath-First-Search($G = (V, E), s, w, d, L$)

while *L is not empty* **do**
  Pop $i \leftarrow L$;
  for *each vertex j where $(i, j) \in E$* **do**
    if $d_{s,i} < d_{s,i}$ **then**
      $b_{s,i} = 1 + \sum_j \sigma_{s,i,j}$;
    end
    if $d_{s,i} > d_{s,j}$ **then**
      $\sigma_{s,i,j} = \frac{w_{s,j}}{w_{s,i}} b_{s,i}$;
    end
  end
end
Return σ;

Girvan-Newman Algorithm

Girvan-Newman($G = (V, E)$)

Calculate edge betweenness for each edge in the graph;
while E *is not empty* **do**
    Let e be the edge with highest edge betweenness;
    $E \leftarrow E - \{e\}$;
    Calculate edge betweenness for remaining edges;
end

Review of Modularity

Definition

Modularity of a network is a scalar ranging from $-\frac{1}{2}$ to 1 that evaluates the density of links inside communities as compared to links between communities.

$$Q = \frac{1}{4m} \sum_{i \in V, j \in V, c_i = c_j} [A_{ij} - \frac{k_i k_j}{2m}]$$

- A_{ij} is the weight of the edge between i and j.
- $k_i = \sum_{(i,j) \in E} A_{ij}$ is the sum of the weights of all the edges attached to vertex i.
- $m = \frac{1}{2} \sum_{ij} A_{ij}$,
- c_i is the community of vertex i.

Girvan-Newman Algorithm

- Girvan-Newman algorithm computes the modularity of current community partition, and determined to terminate when the modularity of the resulting partition reaches a maximum.
- In complex networks it is often the case that more edges have the same highest edge betweenness. We can remove these edges together to effectively reduce the number of iteration.

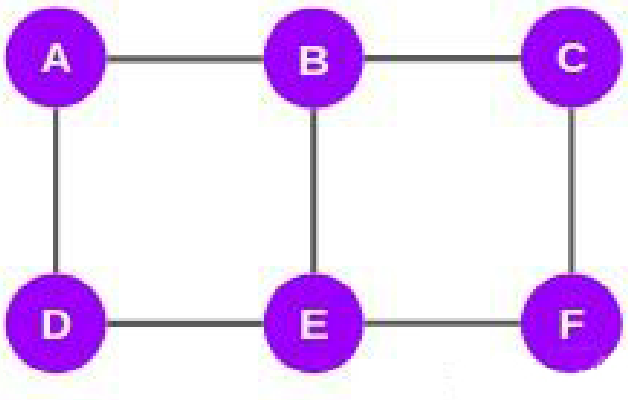
Girvan-Newman Algorithm with Modularity

Modularity-Girvan-Newman($G = (V, E)$)

Calculate edge betweenness for each edge in the graph;
Calculate modularity Q;
while *E is not empty* **do**
    Let E' be the set of all edges with the highest edge betweenness;
    Calculate modularity Q' of $G = (V, E - E')$;
    if $Q' < Q$ **then**
        Break;
    end
    $E \leftarrow E - E'$;
    Calculate edge betweenness for remaining edges;
end
Return(*(V, E)*);

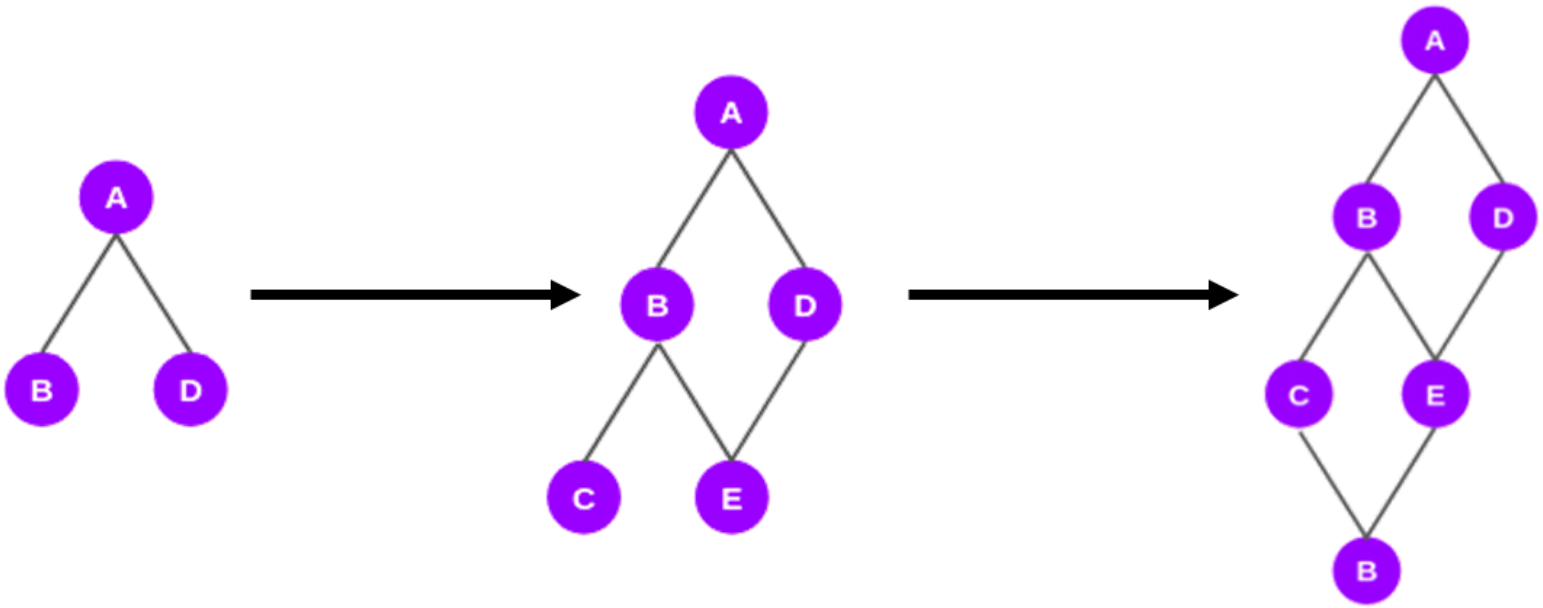
Example

Let's start with a simple network.



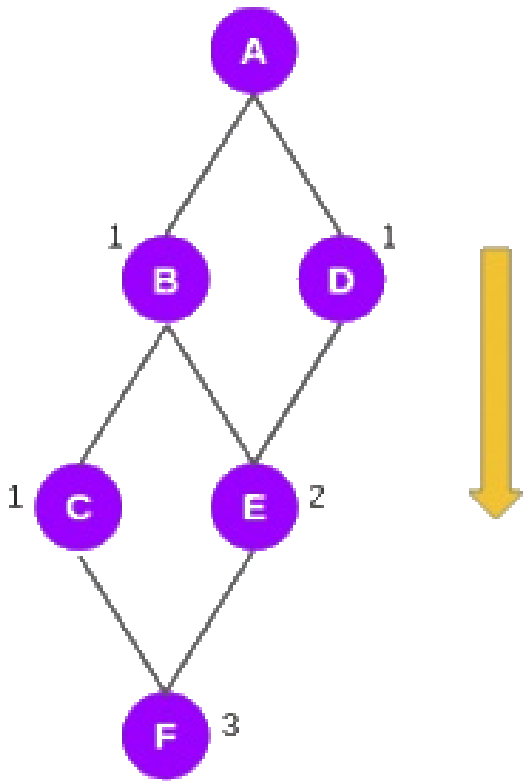
Example

Start from vertex A and BFS the network to get the shortest paths.



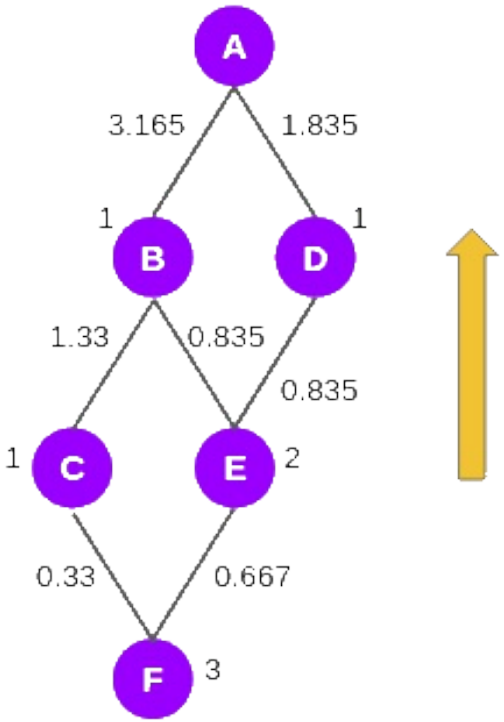
Example

Calculate the number of the shortest paths from vertex A that pass through vertex u.



Example

Reverse BFS to calculate the number of the shortest paths from vertex A that pass through edge e.



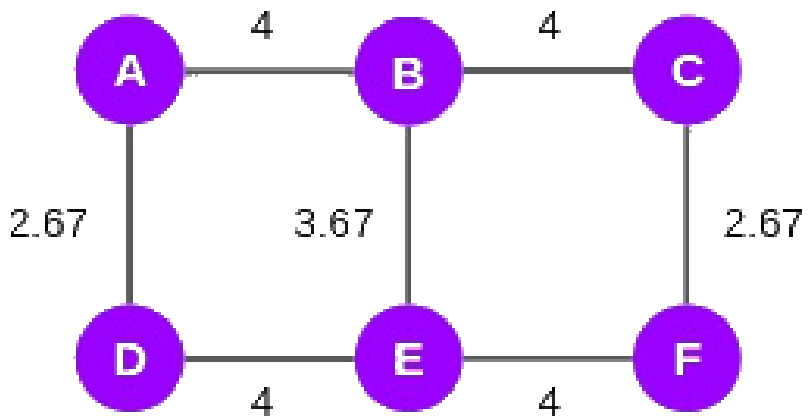
FC = 1/3 = 0.33
FE = 2/3 = 0.667

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

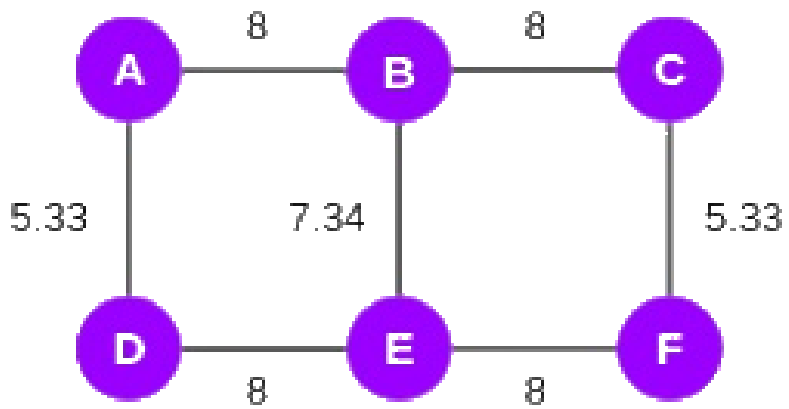
Example

Reverse BFS to calculate the number of the shortest paths from vertex A that pass through edge e.



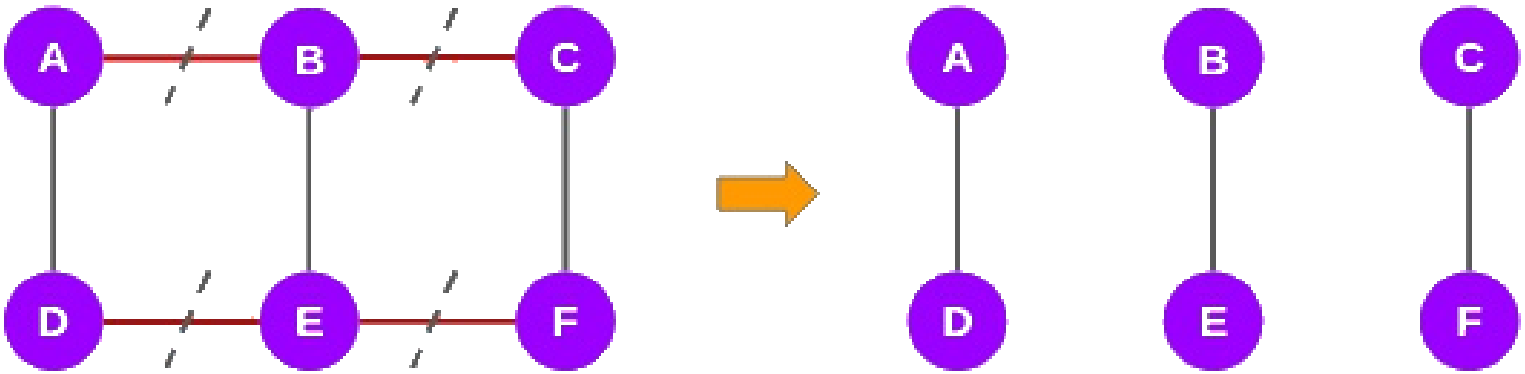
Example

Repeat for each vertex and sum them up.



Example

Remove all edges with the highest edge betweenness.



And then we can get three communities.

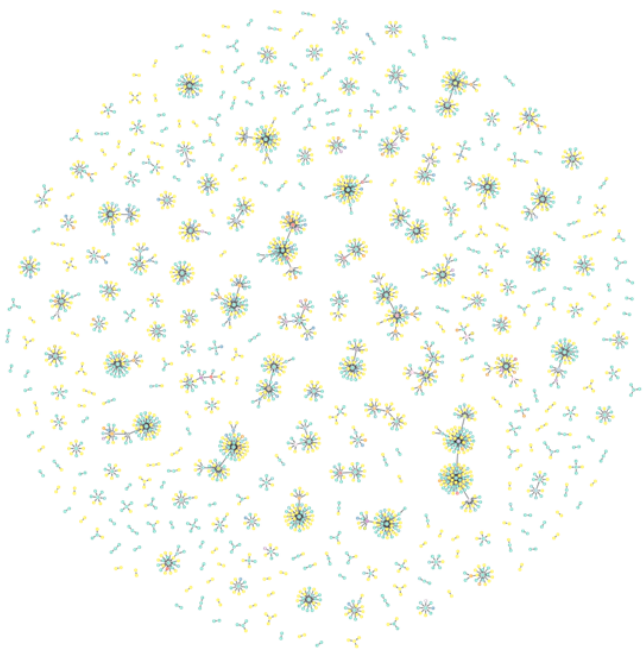
Louvain Algorithm

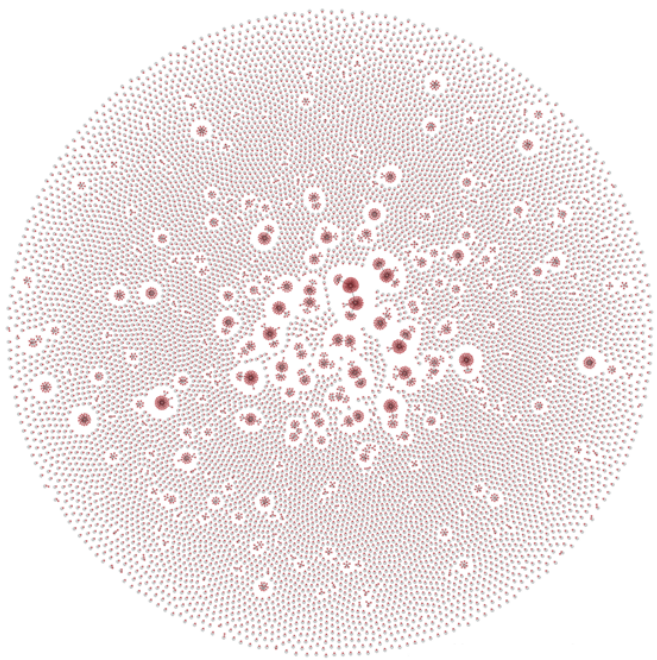
Louvain Algorithm

- Louvain is a multistep technique based on a local optimization of Newman-Girvan modularity in the neighborhood of each node.
- After a partition is identified in this way, communities are replaced by supernodes, yielding a smaller weighted network.
- The procedure is then iterated, until modularity does not increase any further.

Louvain Algorithm

- Louvain is a multistep technique based on a local optimization of Newman-Girvan modularity in the neighborhood of each node.
- After a partition is identified in this way, communities are replaced by supernodes, yielding a smaller weighted network.
- The procedure is then iterated, until modularity does not increase any further.





Modularity Increment

Definition

Modularity increment ΔQ of (i, j) is the change of modularity when removing vertex i from its community and then placing it to the community of its neighbor j.

Modularity Increment

Definition

Modularity increment ΔQ of (i, j) is the change of modularity when removing vertex i from its community and then placing it to the community of its neighbor j.

This removing-placing action can be decomposed into two similar actions:

1) making vertex u isolated
2) moving the isolated vertex into a community

ΔQ is the sum of $\Delta Q'$ of these two actions. These two $\Delta Q'$ can be computed in the same way, for 1) is just the inverse action of 2).

Modularity Increment

$$\Delta Q' = \left[\frac{\sum_{in} + k_{i,in}}{2m} - \left(\frac{\sum_{tot} + k_i}{2m}\right)^2\right] - \left[\frac{\sum_{in}}{2m} - \left(\frac{\sum_{tot}}{2m}\right)^2 - \left(\frac{k_i}{2m}\right)^2\right]$$

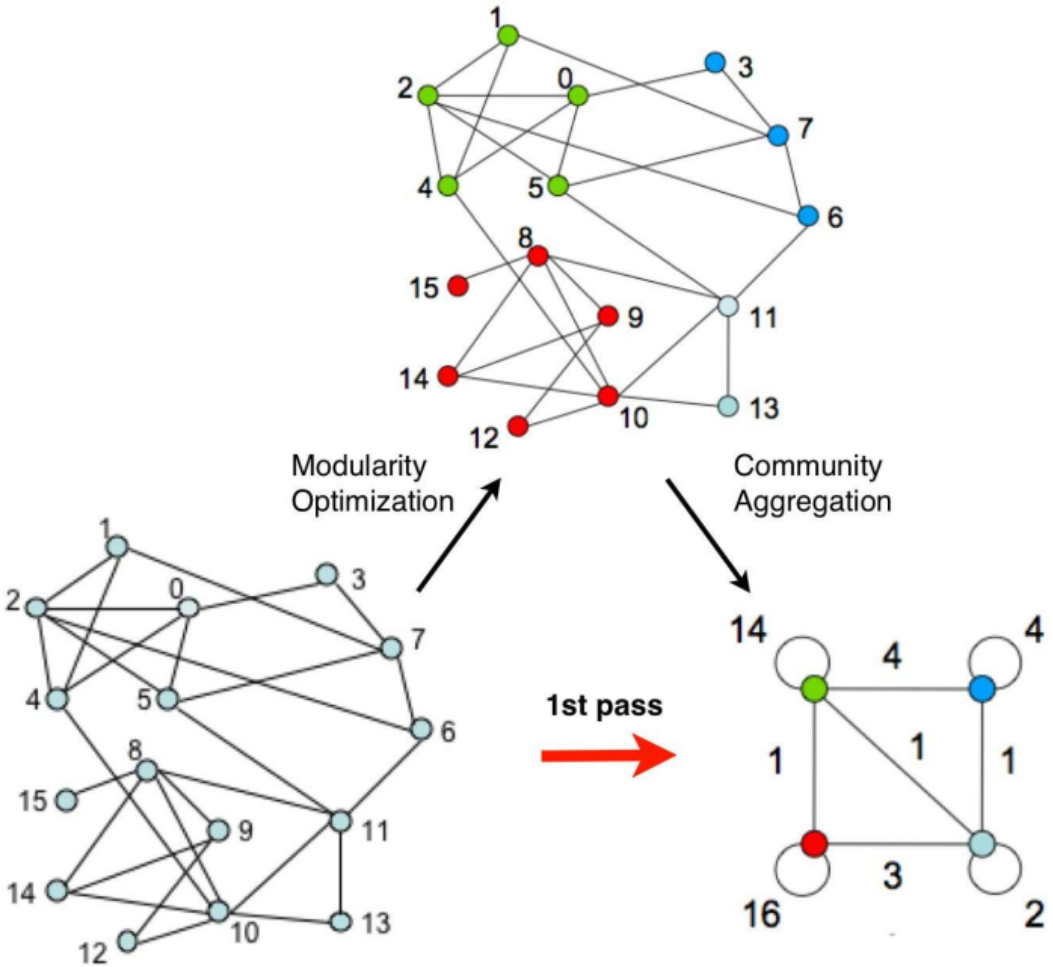
where

- $\sum_{in}$ is the sum of the weights of the internal edges of c_j
- $\sum_{tot}$ is the sum of the weights of the links incident to vertices in community c_j
- $k_{i,in}$ is the sum of the weights of the links from i to vertices in community c_j

Louvain Algorithm

Each iteration, also named pass in Louvain, has two phases.

1) Modularity Optimization
2) Community Aggregation



Modularity Optimization

Modularity-Optimization($G = (V, E)$)

foreach $u \in V$ **do**

  Let v be the neighbor who has the largest modularity increment;

  if *this increment is positive* **then**

    $c_u \leftarrow c$

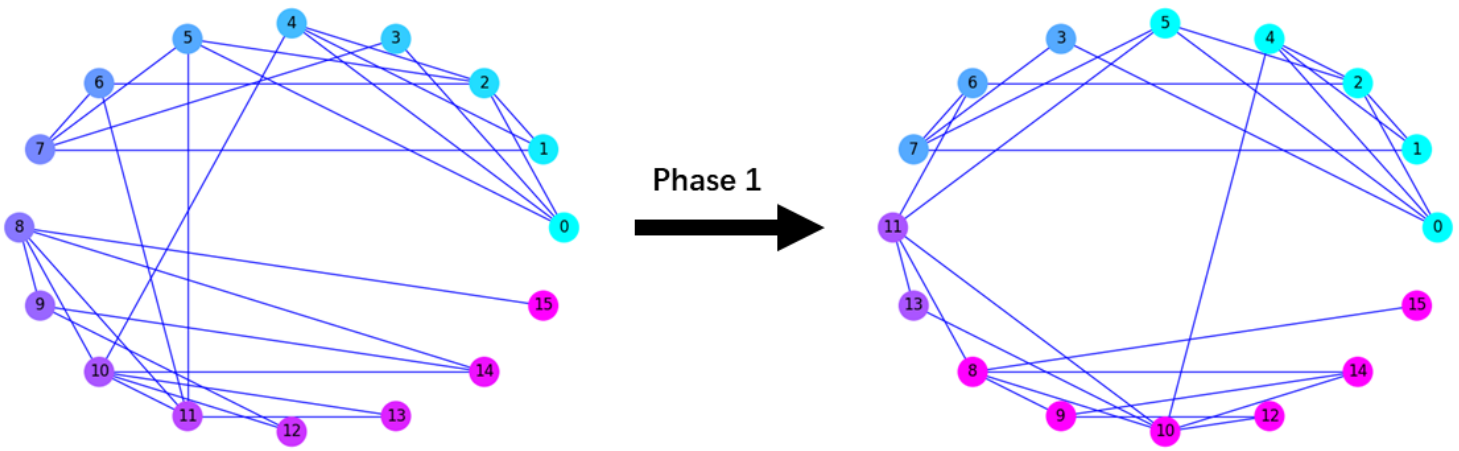
  end

end

Return(c)

Example

Here is the visualization of phase 1.



Community Aggregation

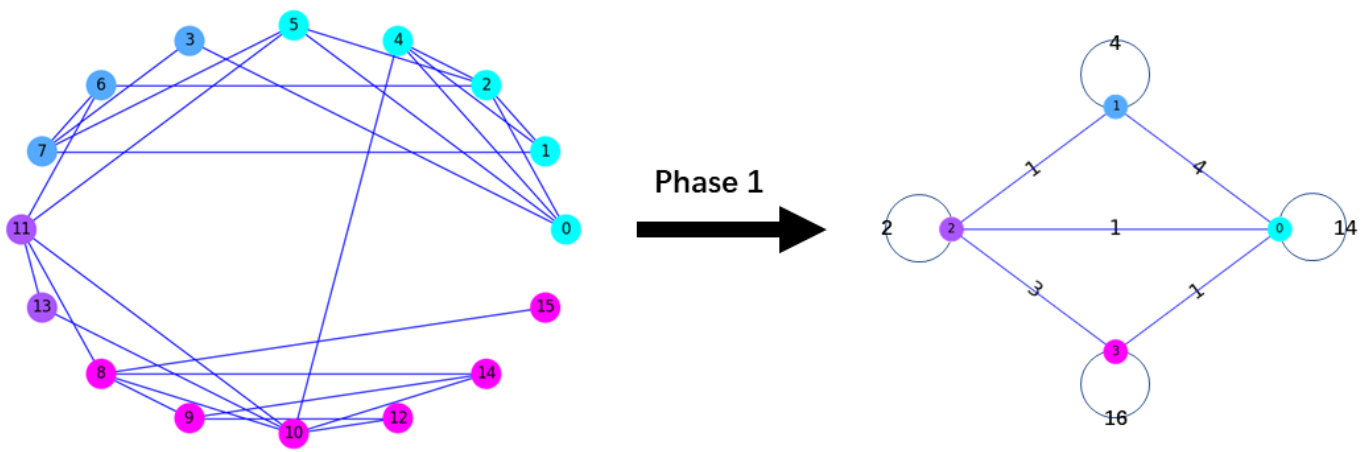
Community-Aggregation($G = (V, E), c$)

foreach *unmerged vertex* u **do**
    foreach *unmerged vertex* $v \in V - \{u\}$ *and* $c_u = c_v$ **do**
        Merge vertex v to u;
    end
    Update internal and external weights of edges;
end
Return(G,c)

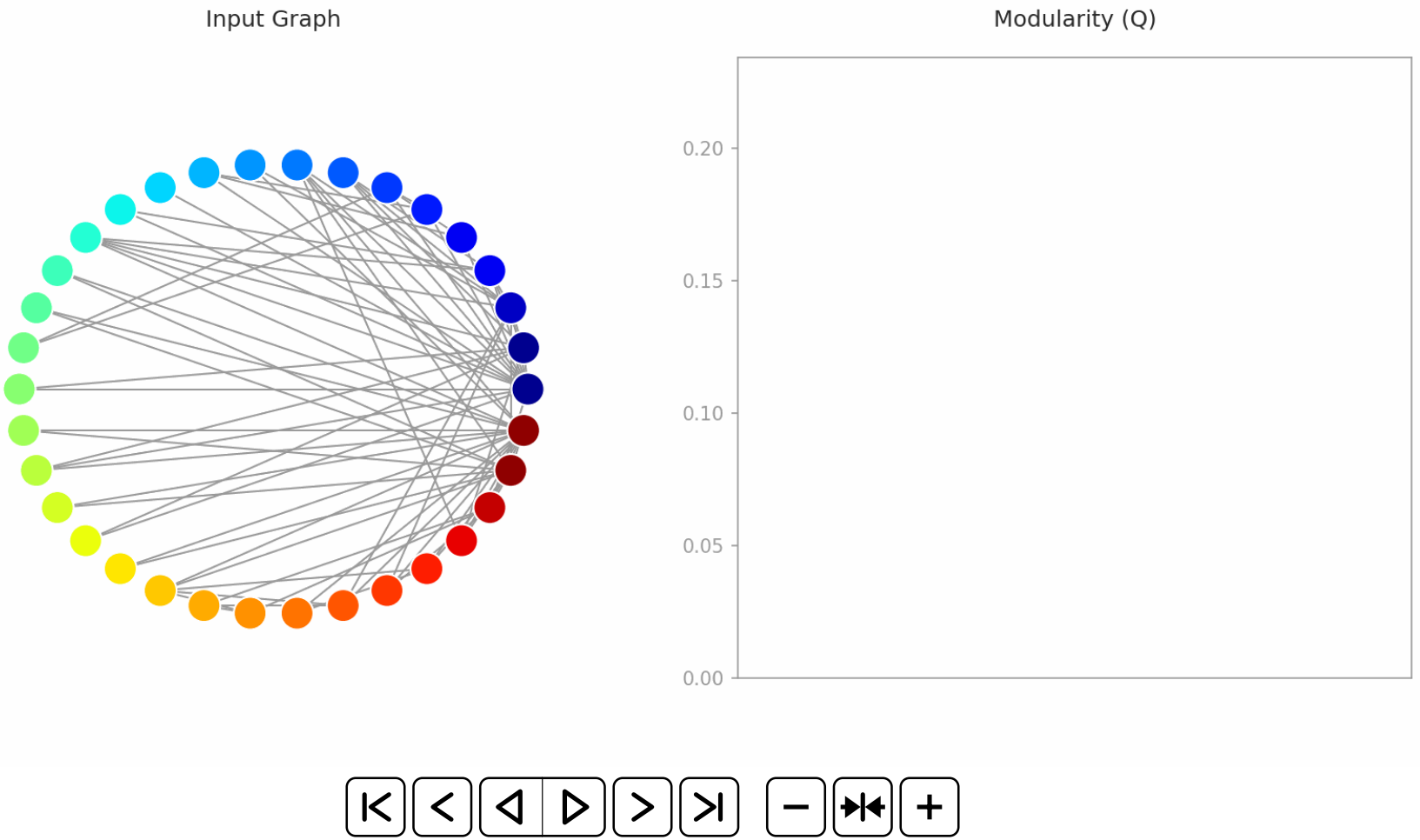
Example

Here is the visualization of phase 2.

- The weights of edges between two new vertices are determined by sum of the weights of the edges between vertices in the corresponding two communities.
- The internal edges of a community leads to a self-loop for the corresponding vertex in the new network.



Visualization of Louvain Algorithm



Parallelization of Louvain Algorithm

Parallelization of Louvain Algorithm

- In the original serial algorithm, each vertex examines the communities of its neighbors and makes a choice to chooses a new community based on a function to maximize the calculated change in modularity.
- In the distributed and parallel version, all vertices make this choice simultaneously rather than in serial order, updating the graph state after each change.

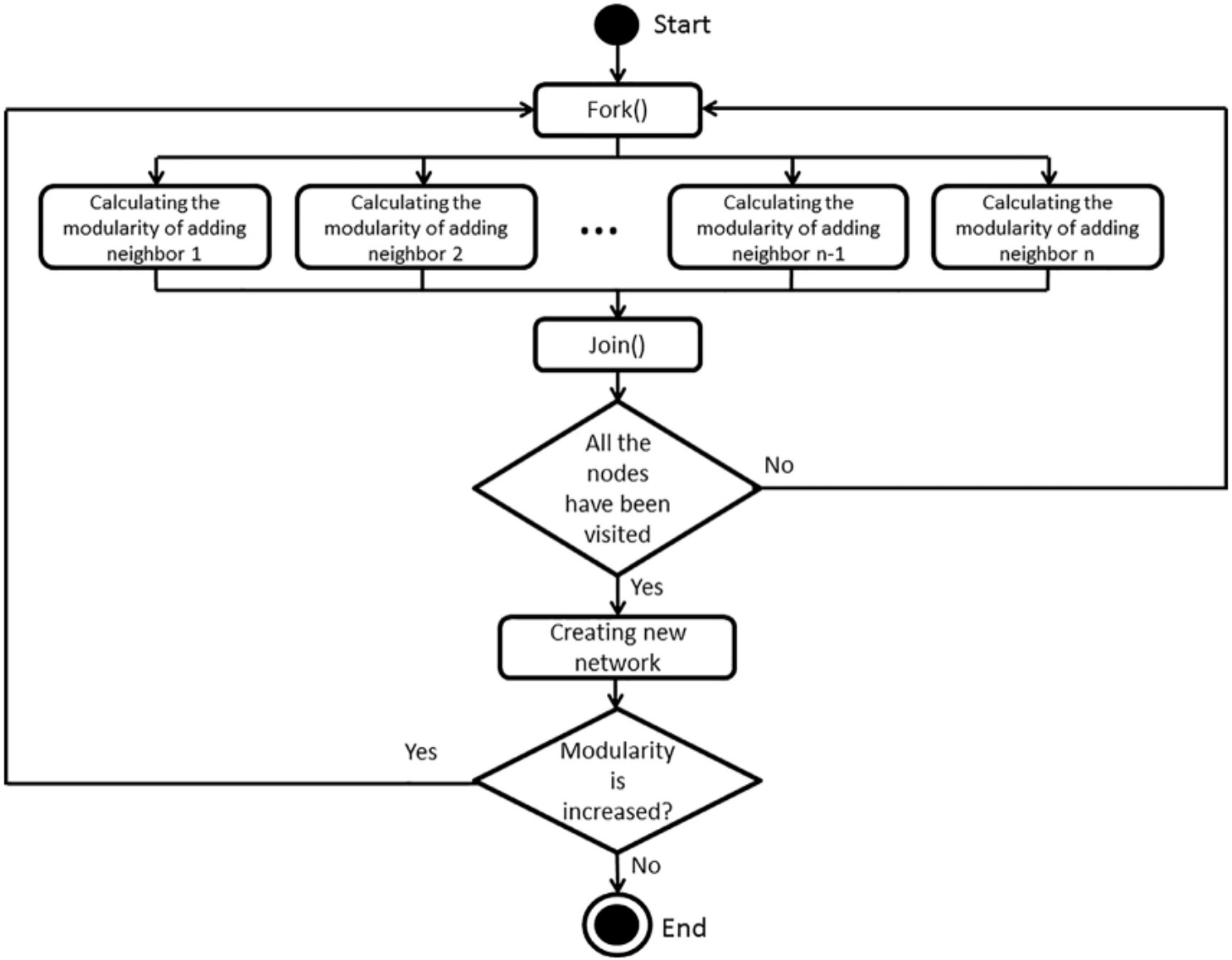
Parallelization of Louvain Algorithm

Parallel-Louvain($G = (V, E)$)

while *True* **do**
  foreach *vertex* $u \in V$ **do**
    Modularity Optimization on vertex u in parallel to get modularity increment ΔQ_u;
  end
  $\Delta Q \leftarrow \sum_{u \in V} \Delta Q_u$;
  if $\Delta Q \leq 0$ **then**
    Break;
  end
  Community-Aggregation(G,c);
end

Parallelization of Louvain Algorithm

The flowchart of the parallel Louvain algorithm:



Parallelization of Louvain Algorithm

Practically, It is a good choice to implement a parallel algorithm on Spark, which is a cluster computing framework supporting reusing a working set of data across multiple parallel operations while retaining the scalability and fault tolerance of MapReduce.

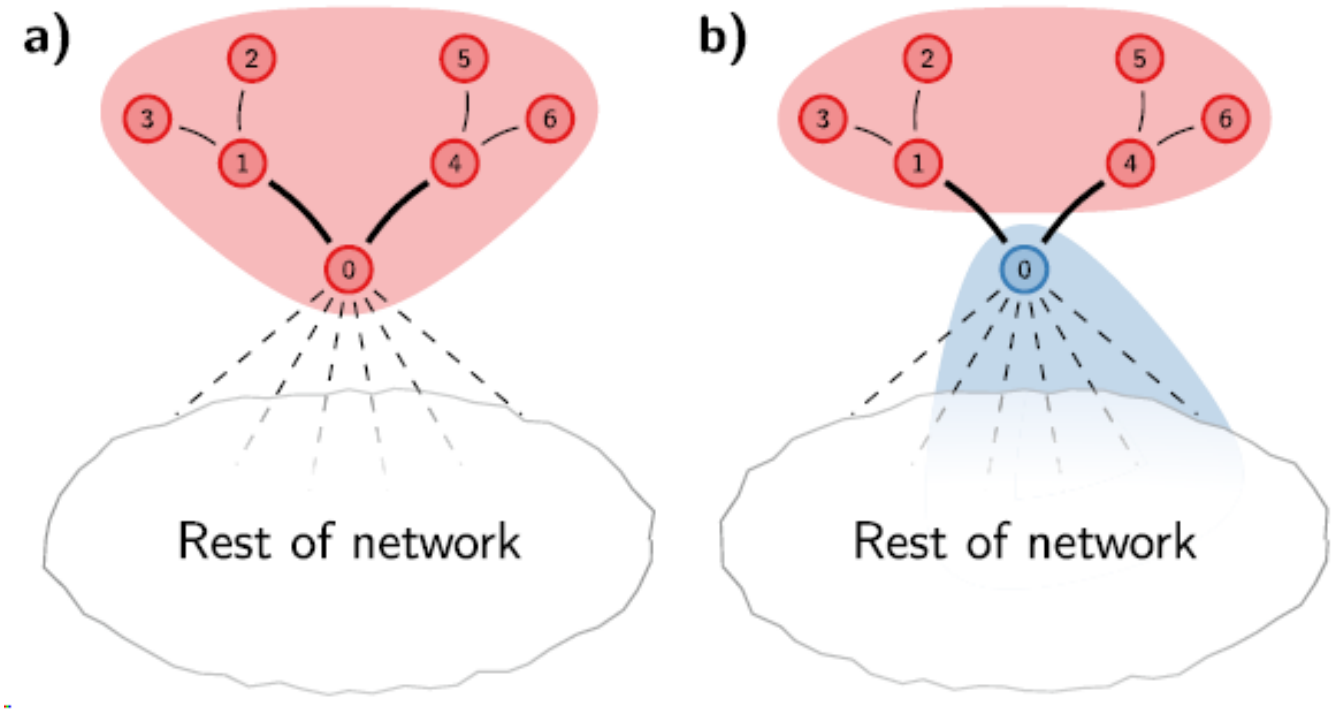
To describe the algorithm using MapReduce framework:

1) Get information of adjacent vertices.
   a) Map: produce information of adjacent vertices VertexData;
   b) Reduce: get information of adjacent vertices (Id, Array[VertexData]).
2) Get the new community (Id, getBestCommunity(Array[VertexData]))
3) Update the information in the network and merge vertices to go to next iteration.

Improvement of Louvain AIgorithm

Drawbacks of Louvain Algorithm

In the Louvain algorithm, a node may be moved to a different community, while it may have acted as a "bridge" between different communities. Removing such a node disconnects the nodes in the old community.

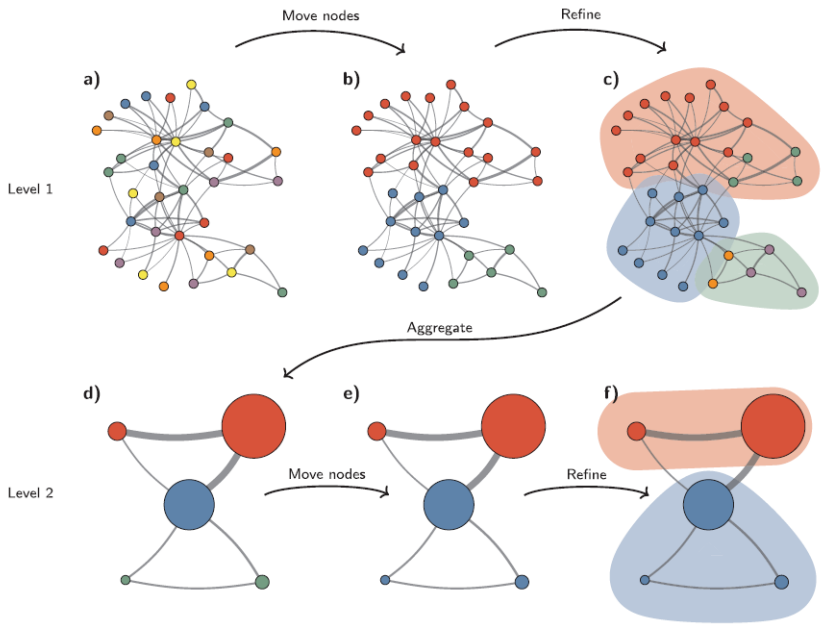


Leiden Algorithm

The Leiden algorithm is more complex than the Louvain algorithm. It consists of three phases:

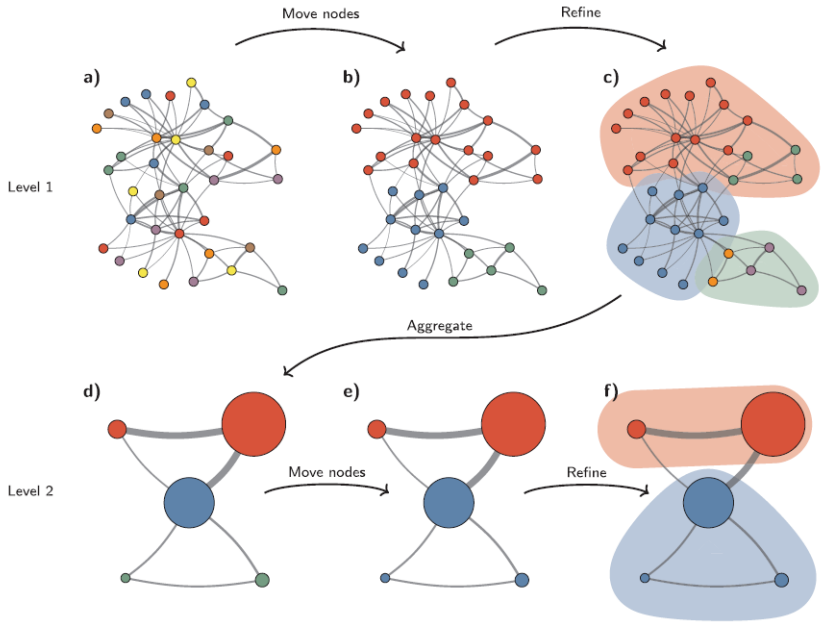
1) Local moving of nodes.
2) Refinement of the partition.
3) Aggregation of the network based on the refined partition, use the non-refined partition to create an initial partition for the aggregate network.

Leiden Algorithm



- The Leiden algorithm starts from a singleton partition (*a*).
- The algorithm moves individual nodes from one community to another to find a partition (*b*), which is then refined (*c*).
- An aggregate network (*d*) is created based on the refined partition, using the non-refined partition to create an initial partition for the aggregate network.

Leiden Algorithm



- The algorithm then moves individual nodes in the aggregate network (*e*).
- In this case, refinement does not change the partition (*f*).
- These steps are repeated until no further improvements can be made.

Comparison Between Louvain and Leiden

Table 1: Comparison between Louvain and Leiden algorithm

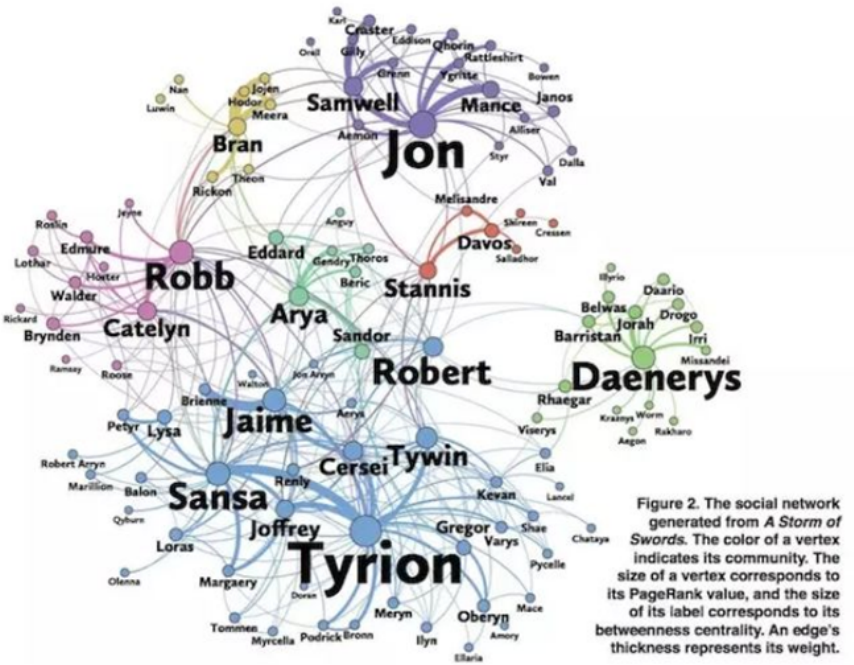
	Louvain	**Leiden**
Year	2008	2019
Proposed by	Blondel et al.	V.A. Traag et al.
Phase	2	3
Advantage(s)	• Performs well on small and medium-scale networks. • Static networks.	• Higher quality function, less time to run. • Temporal networks. • The community are well-connected when running the algorithm iteratively.
Disadvantage(s)	• Communities are badly connected or disconnected, especially when running the algorithm iteratively. • Disable detect in overlapping community.	• Extra step in the process.

Applications of Community Detection

Applications in social networks

An social network is the interaction of people with each other through the web. Community detection has been widely used in this area.

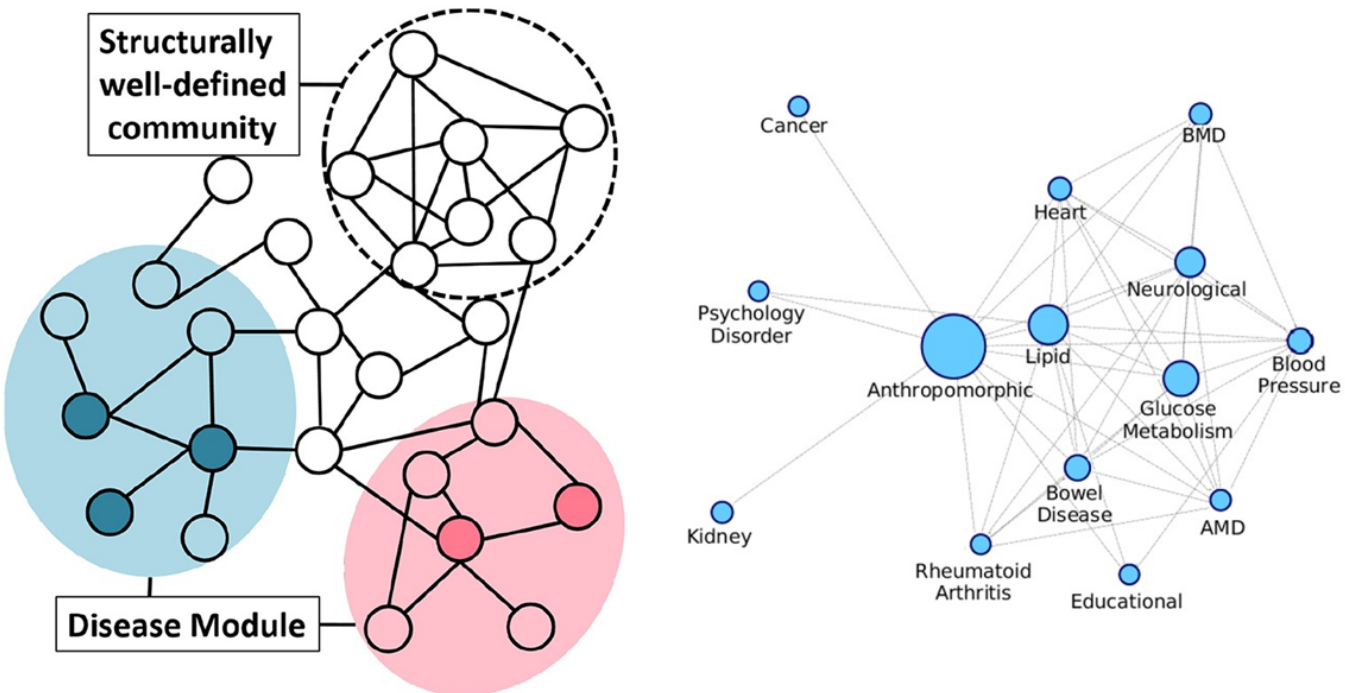
Various studies have utilized different community detection methods to analyze public emotional reaction and visualize relationships and characteristics.



Applications in biological networks

Researchers have proposed an algorithm called disease-gene network detecting algorithm, based on Principal Component Analysis (PCA), which can extract the communities in a diseasome bipartite network.

This algorithm is aimed at disease prevention and medical diagnosis.



Applications in economics

In the stock market, each stock can be represented by a vertex and edge represents the correlations of stock values in the market. Researchers have stated the way to construct the network of stock market and detect communities in it. They revealed community structure by using modularity Q, which helps to the analysis and decision-making of the stock market.

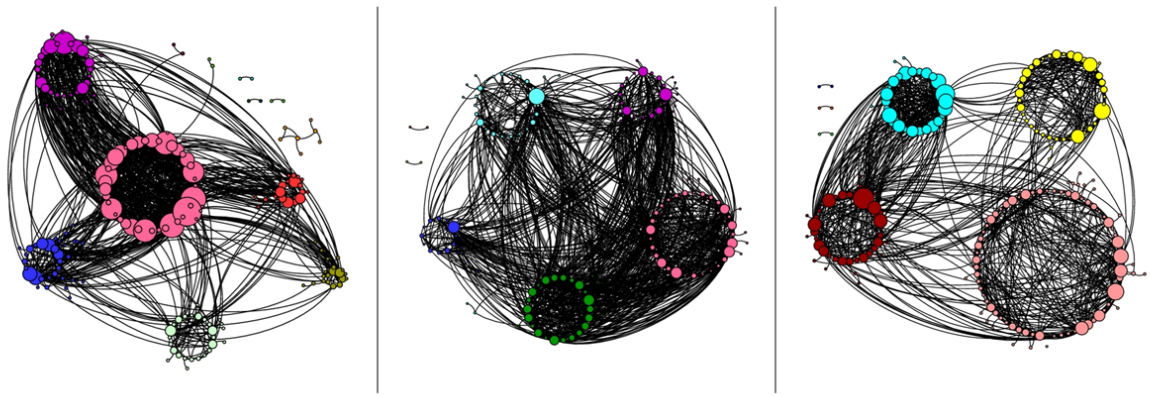
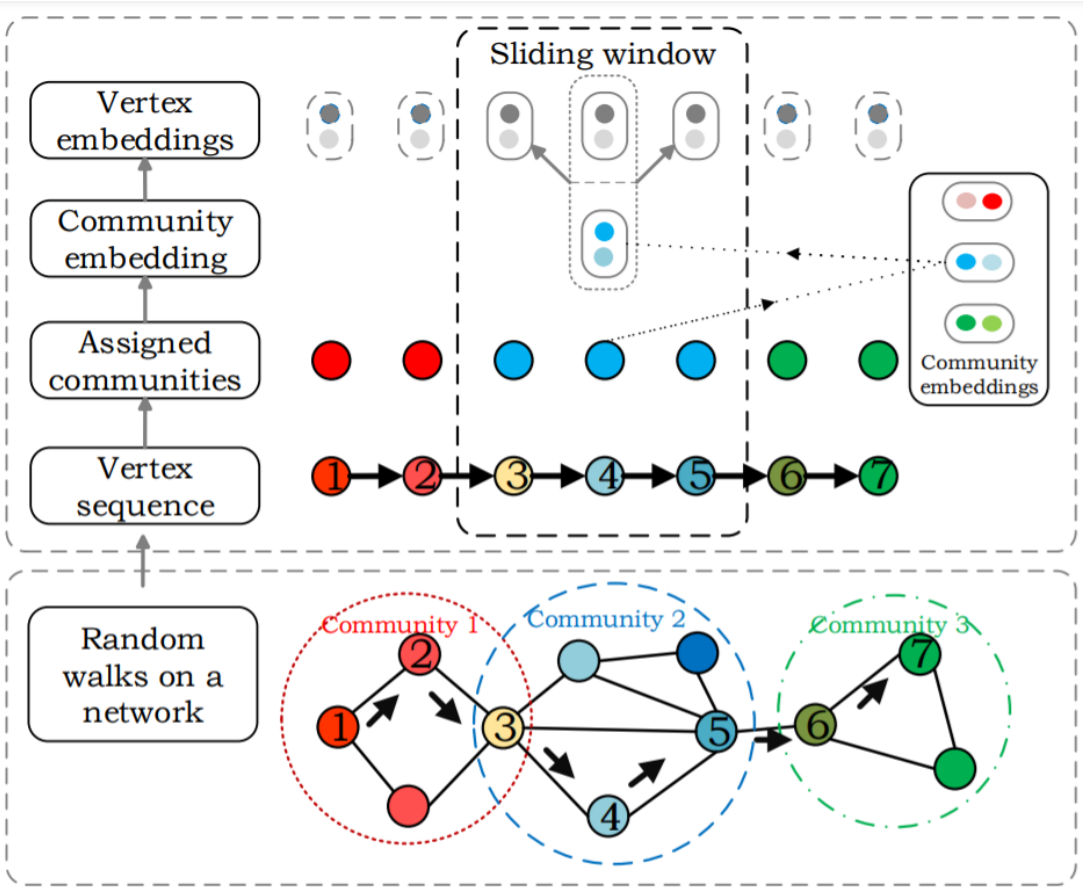


Fig: Several pictures of stock networks. Different node colors represent different communities and node sizes reflect its degree.

Conclusion

Future Direction

Deep learning — a promising direction of community detection. Beyond simply examining network topologies for detecting communities, some strategies also explore semantic descriptions as node features in the data.



Open problems

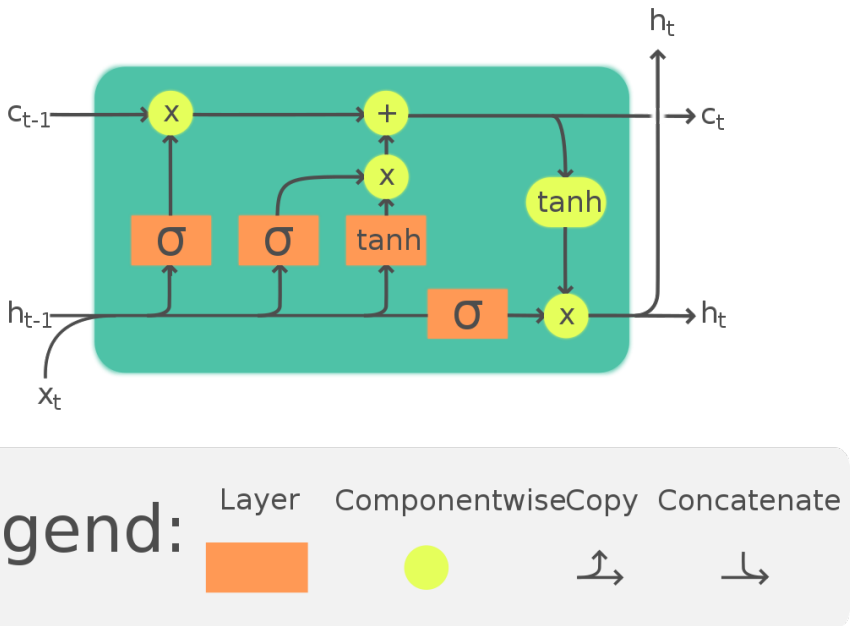
There are still so many broad challenges in community detection. For example:

- Network dynamics
- Large-scale networks
- Inaccurate number of communities

Network dynamics

Changing dynamics can affect either the network topology or the node attributes. Topological changes not only cause changes in a local community, but also leads to devastating changes across an entire network.

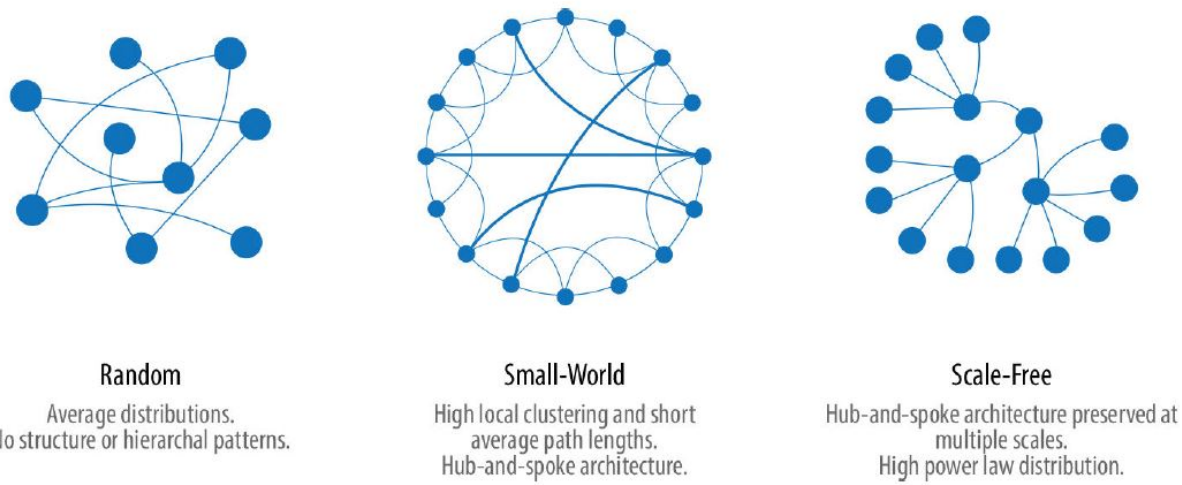
Actually, a lot of methods have been used to deal with sequential data in machine learning, such as long short-term memory (LSTM). Therefore, deep learning methods for detecting communities with dynamic spatial and temporal properties are very likely to be developed.



Large-scale networks

Nowadays, large-scale networks can contain millions of nodes, edges, and structural patterns, as networks like Twitter and Weibo, which has also brought a lot of problems.

For instance, large-scale networks may have their inherent characteristics, such as scale-free. There exists lots of mega hubs in the network, which can influence the performance of algorithms in community detection.



Inaccurate number of communities

In fact, most algorithms of the community detection require the number of communities beforehand as a hyperparameter, including deep learning.

Two common solutions:

- Using modularity-based algorithm, like Girvan-Newman or Louvain
- Using statistical interference

Unfortunately, both methods have poor performance under large-scale networks.