SOCIAL NETWORK ANALYTICS

Community Detection

Prakash C O

Department of Computer Science and Engineering

SOCIAL NETWORK ANALYTICS

Community Detection

What is community structure?
Why Community Detection?
Community Types and Applications

Prakash C O
Department of Computer Science and Engineering

What is Community/Community Structure/Clustering?

- In the context of networks, **community structure** refers to **the occurrence of groups of nodes in a network that are more densely connected internally than with the rest of the network.**

Figure-1 shows a sketch of a small network displaying **community structure**, with three groups of nodes with dense internal connections and sparser connections between groups.

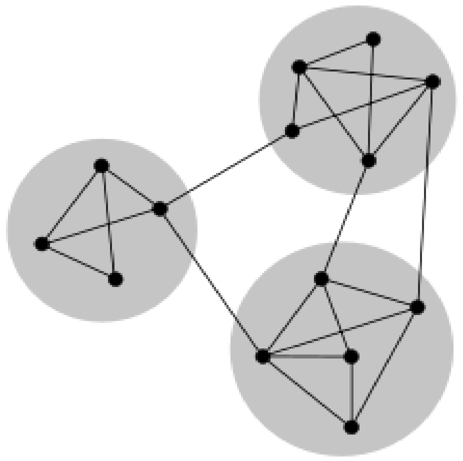
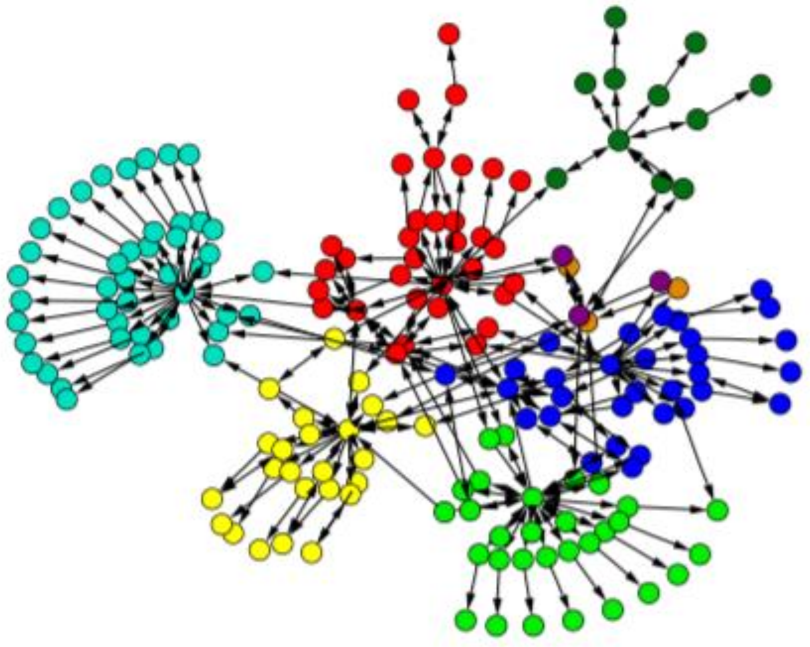


Figure: 1

What is Community/Community Structure/Clustering?

- In real networks, the distribution of edges are locally inhomogeneous, with **high concentrations of edges within special groups of vertices,** and **low concentrations between these groups**.
  This feature of real networks is called community structure (Girvan and Newman, 2002), or clustering.

FIG. 1 **Community structure in technological networks.**
Sample of the web graph consisting of the pages of a web site and their mutual hyperlinks, which are directed.



What is Community/Community Structure/Clustering?

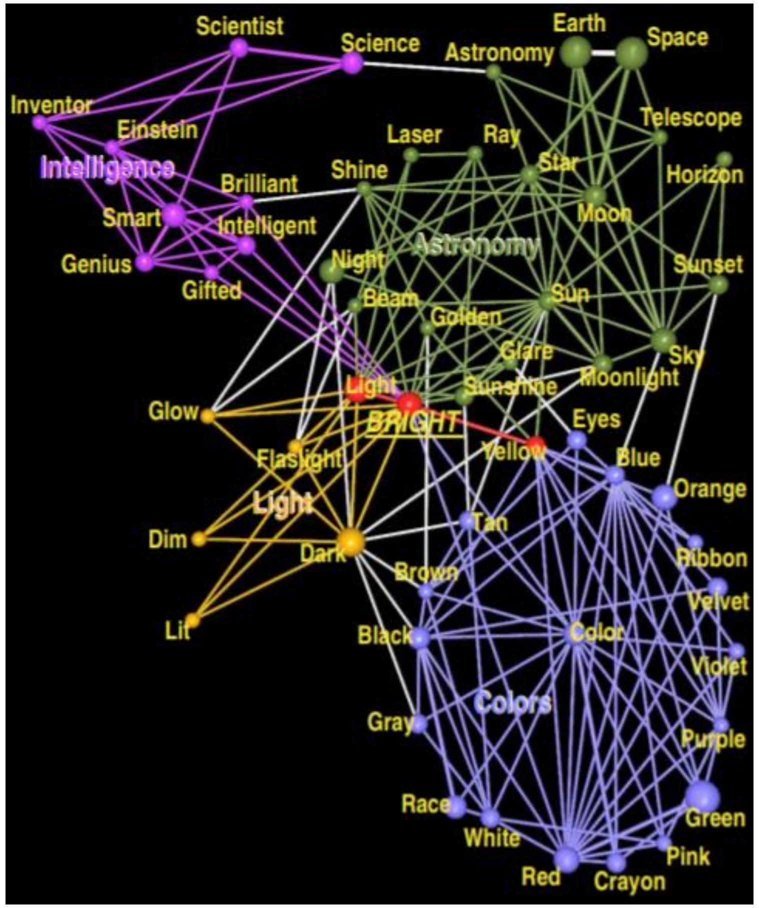


FIG. 2 Overlapping communities in a network of word association.
The groups, labeled by the colors, were detected with the Clique Percolation Method by Palla et al.

SOCIAL NETWORK ANALYTICS

Community Detection

What is Community/Community Structure/Clustering?

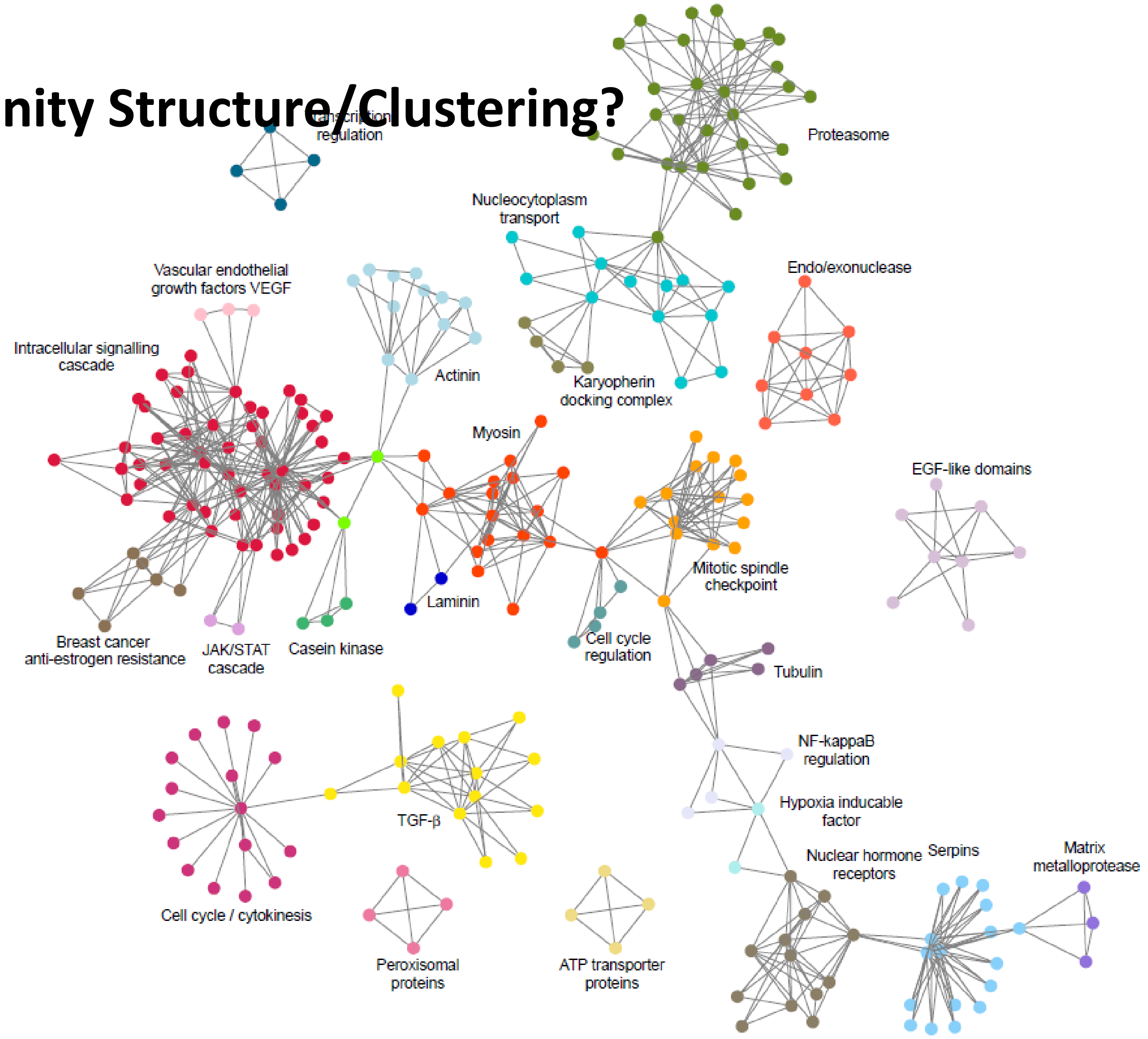
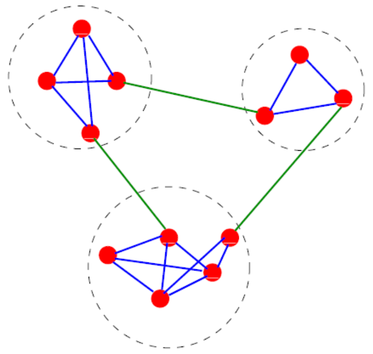


FIG. 3 Community structure in protein-protein interaction networks. The graph pictures the interactions between proteins in cancerous cells of a rat.
Communities, labeled by colors, were detected with the Clique Percolation Method by Palla et al.

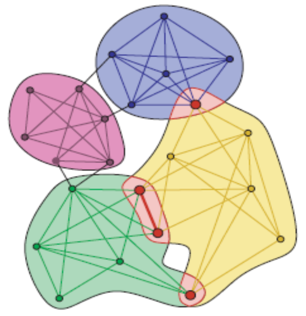
Community Types

➢ **Basically communities can be subdivided into two types:**

1. **Disjoint communities**
   - **In disjoint communities' nodes can be part of only single community**



2. **Overlapped communities**
   - **In overlapped communities' partitions are not bounded to be disjoint.** There could be nodes that belong to multiple communities.



What is Community Detection?

- The concept of community detection has emerged in network science as **a method for finding groups within complex systems.**

- The field of community detection **aims to identify highly connected groups of individuals or objects inside complex networks.**

Why Community Detection?

- As **social networks grew to sizes far beyond the possibility of manual processing**, it became increasingly important to develop computationally efficient and accurate algorithms that can bring important features of a network to the forefront.

- **The identification of communities is essential to understanding of the structure and functionality of large networks.**

Why Community Detection?

- Communities in a citation network might represent related papers on a single topic.

- Communities on the web might represent pages of related topics.

- Clustering Web clients who have similar interests and are geographically near to each other may improve the performance of services provided on the World Wide Web, in that each cluster of clients could be served by a dedicated mirror server (Krishnamurthy and Wang, 2000).

Why Community Detection?

- Identifying clusters of customers with similar interests in the network of purchase relationships between customers and products of online retailers (like, e. g., www.amazon.com) enables to set up efficient recommendation systems (Reddy et al., 2002), that better guide customers through the list of items of the retailer and enhance the business applications
- Clusters of large graphs can be used to create data structures in order to efficiently store the graph data and to handle navigational queries, like path searches (Agrawal and Jagadish, 1994; Wu et al., 2004).
- Community can be considered as a summary of the whole network thus easy to visualize and understand.

Community structure: Applications

- Community structures are quite common in real networks.
  - **Social networks** include community groups based on common location, interests, occupation, etc.
  - **Metabolic networks** have communities based on functional groupings.
  - **Citation networks** form communities by research topic.
  - **Mobile phone communication networks** have communities based on ……
- Being able to **identify these community structures within a network** can **provide insight into how network function** and **topology affect each other**.
- The **ability to find and analyze such community structures** can **provide invaluable help in understanding and visualizing the structure of networks.**

Communities in social media sites

➢ Examples of explicit communities in well-known social media sites include the following:

- **Facebook.** In Facebook, there exist a variety of explicit communities, such as groups and communities. **In these communities, users can post messages and images, comment on other messages**, like posts, and view activities of others.
- **Yahoo! Groups/Google groups**. In Yahoo! groups, **individuals join a group mailing list** where they can receive emails from all or a selection of group members (administrators) directly.
- **LinkedIn.** LinkedIn provides its users with a feature called Groups and Associations. **Users can join professional groups where they can post and share information related to the group**.

Community Detection

➢ **Algorithms for finding communities**

1. **Girvan–Newman algorithm**
2. **The Louvain algorithm**
3. **Hierarchical clustering**
4. **Clique based methods**
5. Minimum-cut method
6. Modularity maximization
7. Statistical inference

Community Detection

Basic Idea

- A simple divisive strategy:
  - Repeat
    1. Find out one “inter-community” edge.
    2. Remove the edge
    3. Check if there’s any disconnected components (which corresponds to a community)

How to measure “inter-community”

- If two communities are joined by a few inter-community edges, then all the paths from one community to another must pass the edges.
- **Various measures:**
  1. **Edge Betweenness**: find the shortest paths between all pairs of nodes and count how many run along each edge.
  2. **Random Walk betweenness**.
  3. **Current-flow betweenness**

SOCIAL NETWORK ANALYTICS

Community Detection

Girvan-Newman Algorithm

Prakash C O

Department of Computer Science and Engineering

Community Detection: Girvan–Newman

Girvan–Newman algorithm.

a) Calculate the betweenness for all edges in the network.
   - find the shortest paths between all pairs of vertices and count how many run along each edge.

b) Remove the edge with the highest betweenness.

c) Recalculate betweennesses for all edges affected by the removal.

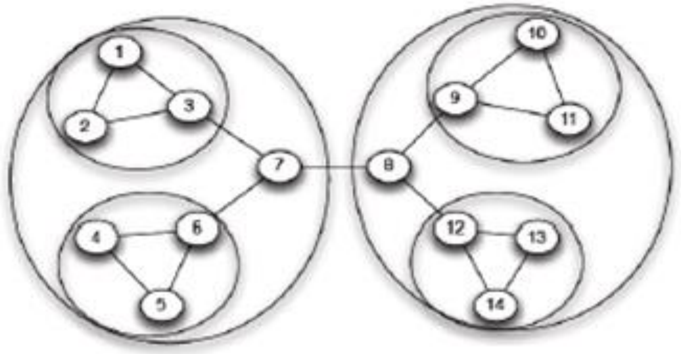
d) Repeat from step 2 until no edges remain.

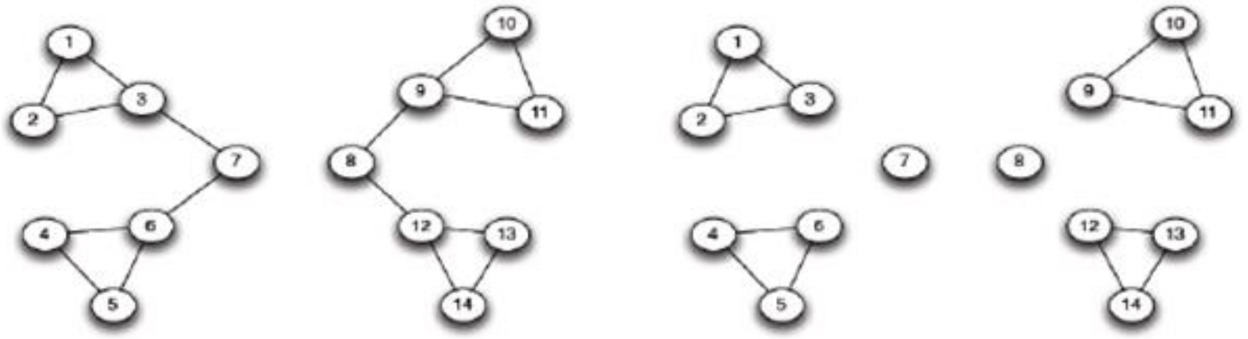
Note: Edge betweenness is the number of shortest paths between all vertex pairs that run along the edge.

Community Detection: Girvan–Newman

➢Girvan–Newman algorithm: **Example**

successively remove edges of highest betweenness (the bridges, or local bridges), breaking up the network into separate components





(a) *Step 1* (b) *Step 2*

Community Detection: Girvan–Newman

- The Girvan–Newman algorithm returns results of reasonable quality and is **popular because it has been implemented in a number of standard software packages**.

- But it also runs slowly, **taking time O(m^2n) on a network of n vertices and m edges**, making it impractical for networks of more than a few thousand nodes

Community Detection: Girvan–Newman

Girvan–Newman algorithm.

a) Calculate the betweenness for all edges in the network.
   - find the shortest paths between all pairs of vertices and count how many run along each edge.

b) Remove the edge with the highest betweenness.

c) Recalculate betweennesses for all edges affected by the removal.

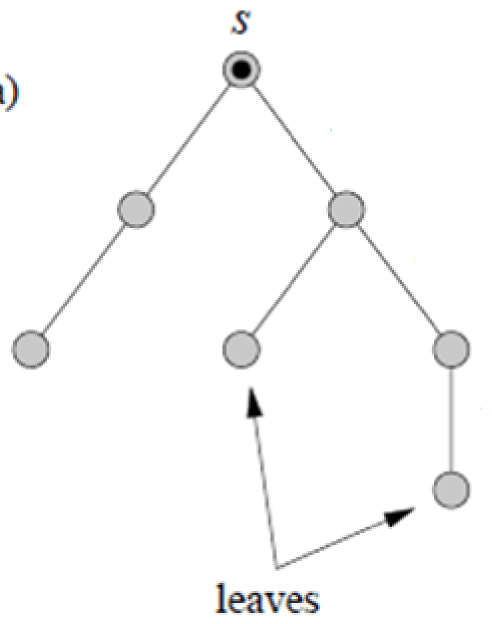
d) Repeat from step 2 until no edges remain.

Community Detection: Girvan–Newman

1. Breadth-first search can find shortest paths from a single vertex s to all others in time O(m).

In the simplest case, when there is only a single shortest path from the source vertex to any other the resulting set of paths forms a shortest path tree—see Fig. a.

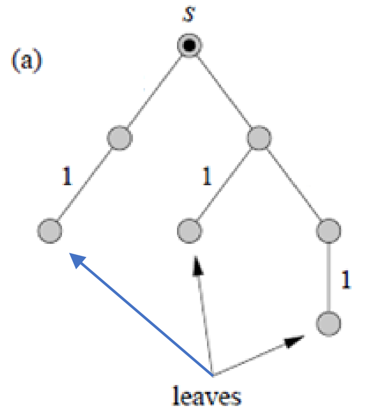
(a)

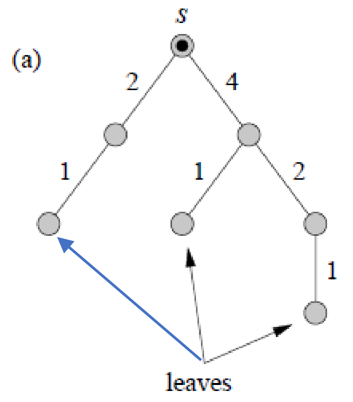


Community Detection: Girvan–Newman

2. Now use this tree to **calculate the contribution to betweenness for each edge from this set of paths** as follows.

- Find first the "**leaves**" of the tree, i.e., those **nodes such that no shortest paths to other nodes pass through them**, and assign a score of 1 to the single edge that connects each to the rest of the tree, as shown in the figure.
- Then, **starting with those edges that are farthest from the source vertex** on the tree, i.e., lowest in Fig. a, we work upwards, **assigning a score to each edge that is 1 plus the sum of the scores on the neighboring edges immediately below it.**
- When we have gone though all edges in the tree, the resulting scores are the betweenness counts for the paths from vertex s.
- Repeating the process for all possible vertices s and summing the scores, we arrive at the full betweenness scores for shortest paths between all pairs.



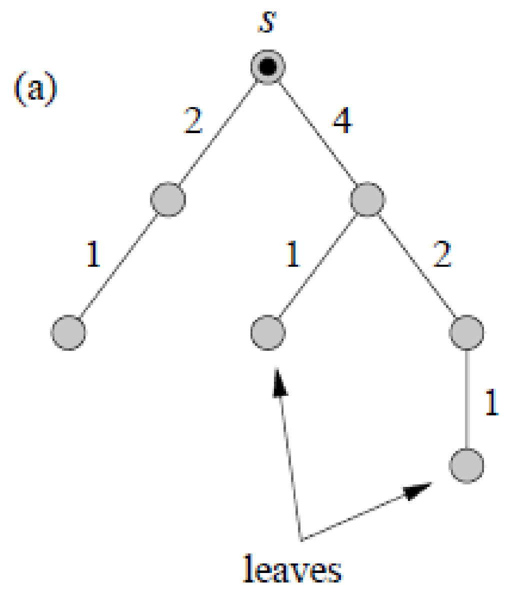


Community Detection: Girvan–Newman

Calculation of shortest-path betweenness:

- **(a) When there is only a single shortest path from a source vertex s (top) to all other reachable vertices**, those paths necessarily form a tree (Figure below)

Assigning a score to each edge that is 1 plus the sum of the scores on the neighboring edges immediately below it.



Community Detection: Girvan–Newman

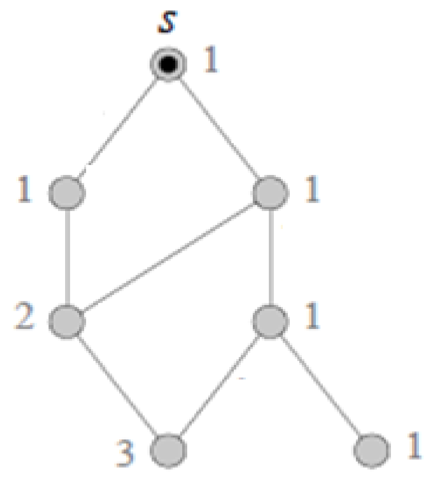
Calculation of shortest-path betweenness:

- **(b)** For cases in which **there is more than one shortest path to some vertices, the calculation is more complex**.

  First we must **calculate the number of paths from the source to each other vertex** (numbers on vertices), and **then these are used to weight the path counts appropriately.**

  The weight on a vertex i represents the number of distinct shortest paths from the source vertex to i.

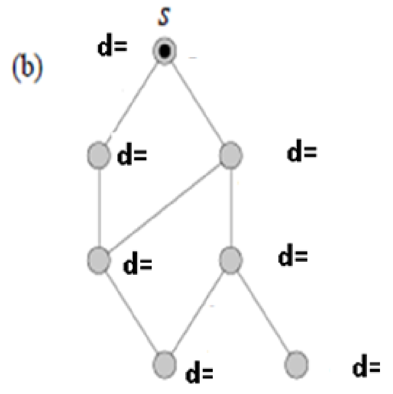
(b)

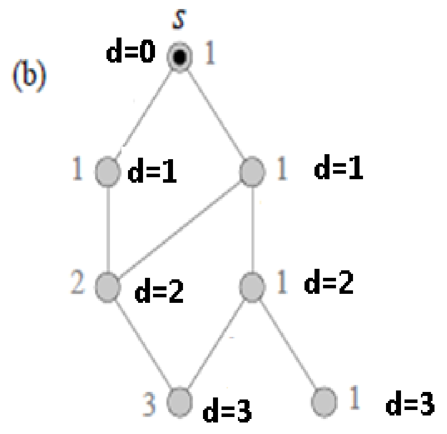


Community Detection: Girvan–Newman

Calculation of shortest-path betweenness: suppose we are performing a breadth-first search starting at vertex s. We carry out the following steps:

1. The initial vertex s is given distance $d_s = 0$ and a weight $w_s = 1$.
2. Every vertex i adjacent to s is given distance $d_i = d_s + 1 = 1$, and weight $w_i = w_s = 1$.
3. For each vertex j adjacent to one of *those* vertices i we do one of three things:
   (a) If j has not yet been assigned a distance, it is assigned distance $d_j = d_i + 1$ and weight $w_j = w_i$.
   (b) If j has already been assigned a distance and $d_j == (d_i + 1)$ then the vertex's weight is increased by w_i, that is $w_j \leftarrow w_j + w_i$.
   (c) If j has already been assigned a distance and $d_j < d_i + 1$, we do nothing.
4. Repeat from step 3 until no vertices remain that have assigned distances but whose neighbors do not have assigned distances.

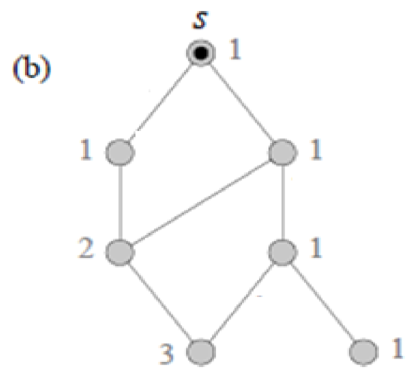




Community Detection: Girvan–Newman

Calculation of shortest-path betweenness:

Physically, the **weight on a vertex i represents the number of distinct paths from the source vertex to i**. These weights are precisely what we need to calculate our edge betweennesses, because **if two vertices i and j are connected, with j farther than i from the source s**, then **the fraction of a geodesic path from j through i to s is given by w_i/w_j .**

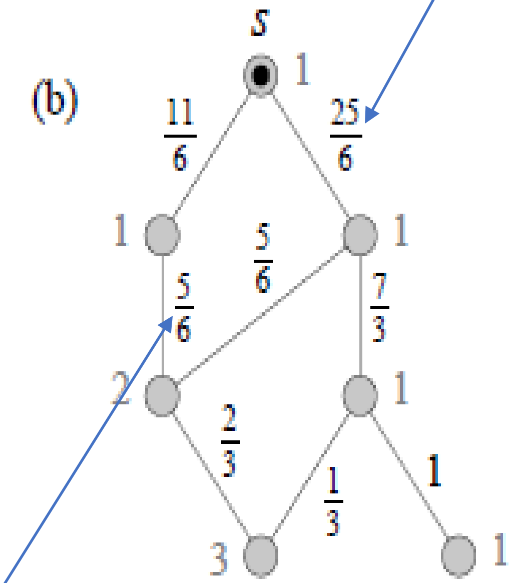


Community Detection: Girvan–Newman

To calculate the contribution to edge betweenness from all shortest paths starting at s, we need to carry out the following steps:

1. Find every “leaf” vertex t, i.e., a vertex such that no paths from s to other vertices go though t.
2. For each vertex i neighboring t assign a score to the edge from t to i of w_i/w_t.
3. Now, starting with the edges that are farthest from the source vertex s—lower down in a diagram such as Fig. 4b—work up towards s. To the edge from vertex i to vertex j, with j being farther from s than i, assign a score that is 1 plus the sum of the scores on the neighboring edges immediately below it (i.e., those with which it shares a common vertex), all multiplied by w_i/w_j.
4. Repeat from step 3 until vertex s is reached.

$\{1+(5/6)+(7/3)\} * (1/1) = 25/6$



$\{1+(2/3)\} * (1/2) = 5/6$

Now repeating this process for all *n* source vertices *s* and summing the resulting scores on the edges gives us the total betweenness for all edges in time O(mn).

Community Detection: Girvan–Newman

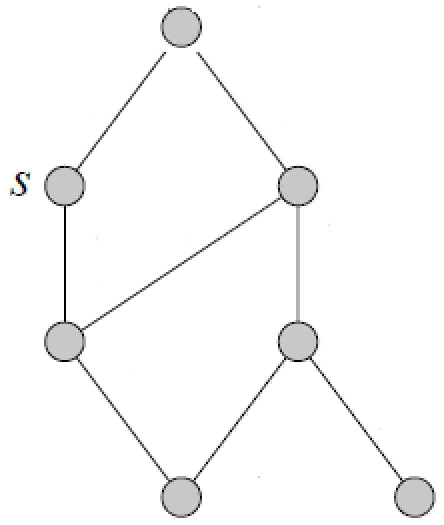
Calculation of shortest-path betweenness:

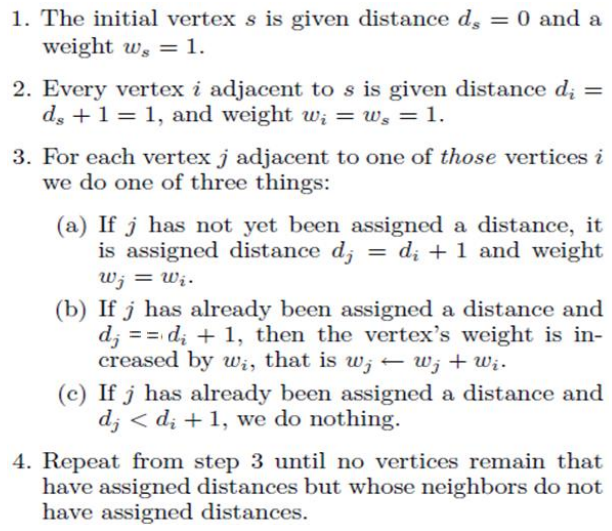
- **Repeat this calculation for each edge removed from the network**, of which there are m, and hence **the complete community structure algorithm based on shortest-path betweenness operates in worst-case time $O(m^2n)$, or $O(n^3)$ time on a sparse graph**.

Community Detection: Girvan–Newman

Calculation of shortest-path betweenness:

Calculate the contribution to edge betweenness from all shortest paths starting at s.



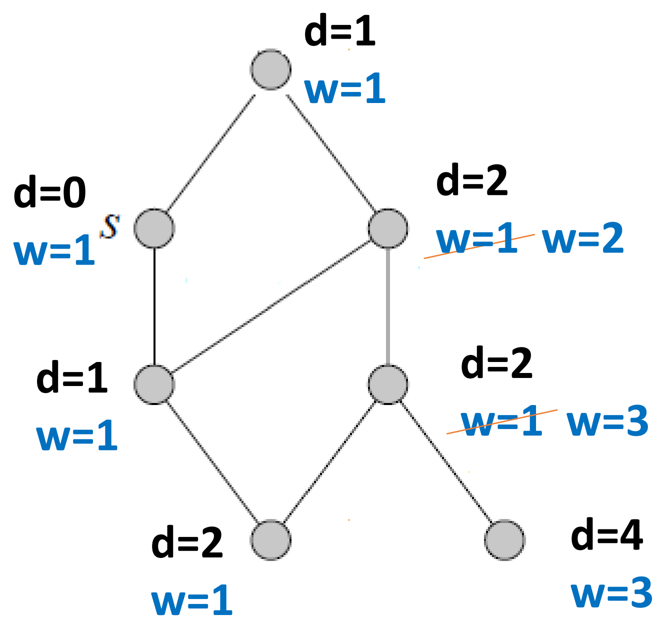


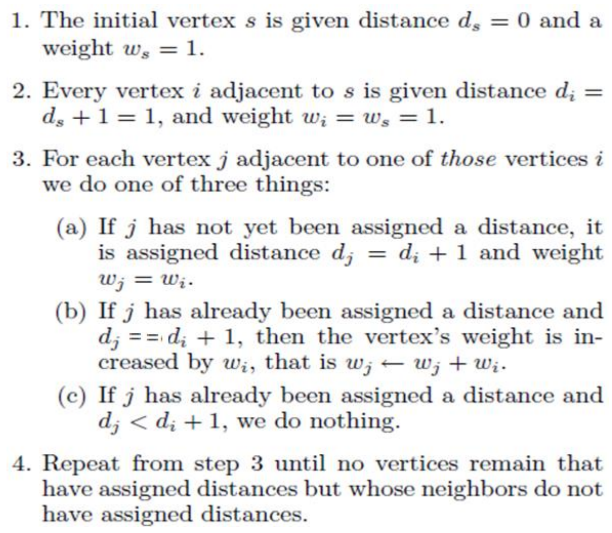
1. The initial vertex s is given distance $d_s = 0$ and a weight $w_s = 1$.
2. Every vertex i adjacent to s is given distance $d_i = d_s + 1 = 1$, and weight $w_i = w_s = 1$.
3. For each vertex j adjacent to one of *those* vertices i we do one of three things:
   - (a) If j has not yet been assigned a distance, it is assigned distance $d_j = d_i + 1$ and weight $w_j = w_i$.
   - (b) If j has already been assigned a distance and $d_j == d_i + 1$, then the vertex's weight is increased by w_i, that is $w_j \leftarrow w_j + w_i$.
   - (c) If j has already been assigned a distance and $d_j < d_i + 1$, we do nothing.
4. Repeat from step 3 until no vertices remain that have assigned distances but whose neighbors do not have assigned distances.

Community Detection: Girvan–Newman

Calculation of shortest-path betweenness:

Calculate the contribution to edge betweenness from all shortest paths starting at s.



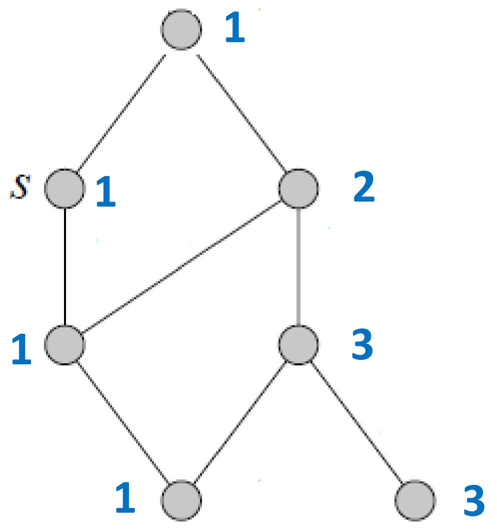


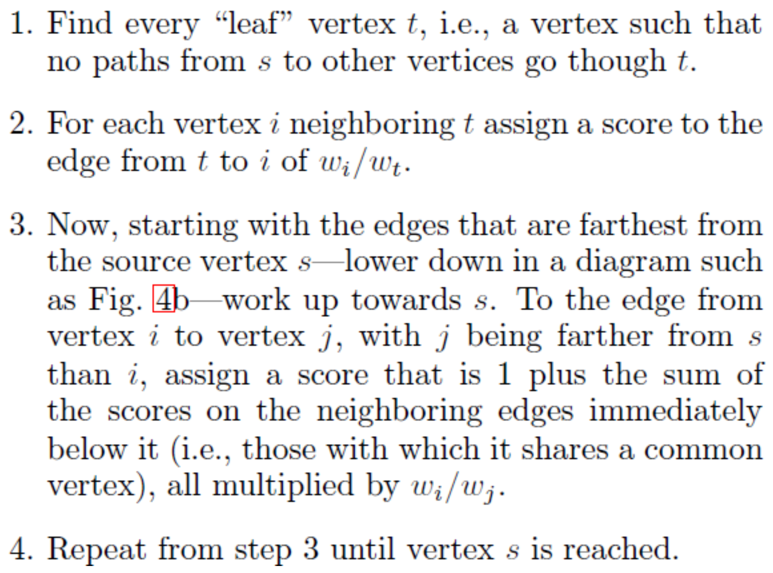
1. The initial vertex s is given distance $d_s = 0$ and a weight $w_s = 1$.
2. Every vertex i adjacent to s is given distance $d_i = d_s + 1 = 1$, and weight $w_i = w_s = 1$.
3. For each vertex j adjacent to one of *those* vertices i we do one of three things:
   (a) If j has not yet been assigned a distance, it is assigned distance $d_j = d_i + 1$ and weight $w_j = w_i$.
   (b) If j has already been assigned a distance and $d_j == d_i + 1$, then the vertex's weight is increased by w_i, that is $w_j \leftarrow w_j + w_i$.
   (c) If j has already been assigned a distance and $d_j < d_i + 1$, we do nothing.
4. Repeat from step 3 until no vertices remain that have assigned distances but whose neighbors do not have assigned distances.

Community Detection: Girvan–Newman

Calculation of shortest-path betweenness:

Calculate the contribution to edge betweenness from all shortest paths starting at s.

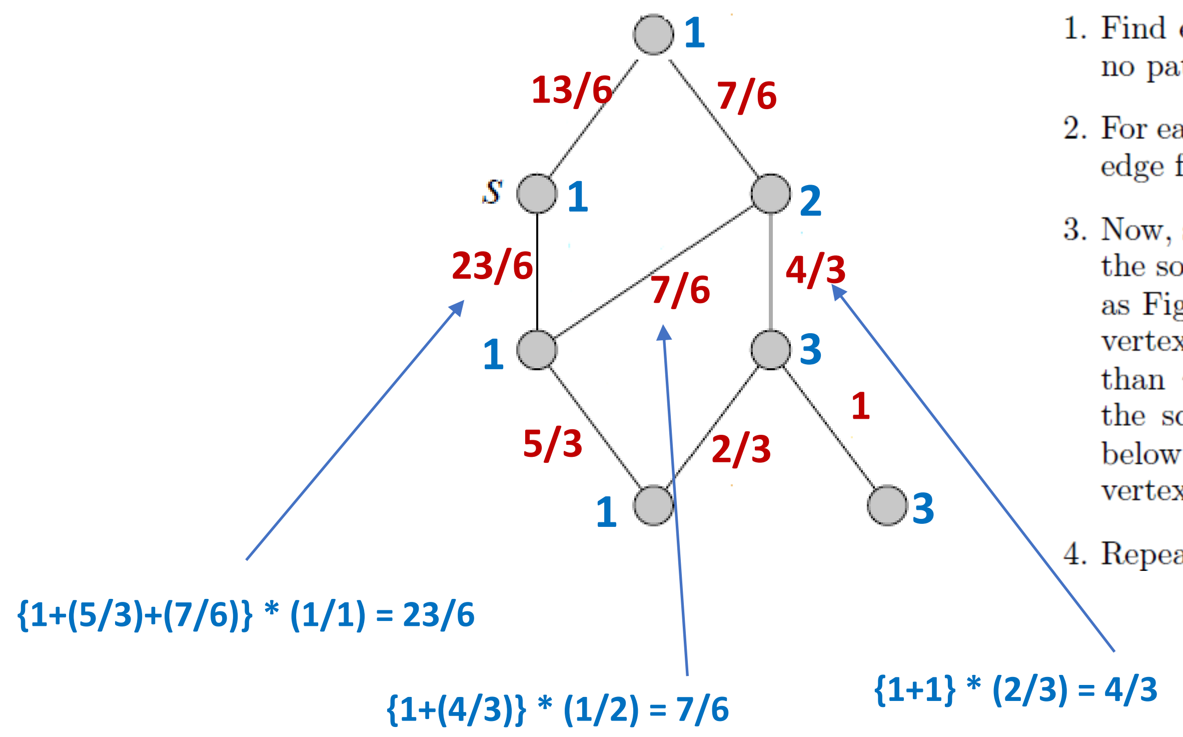


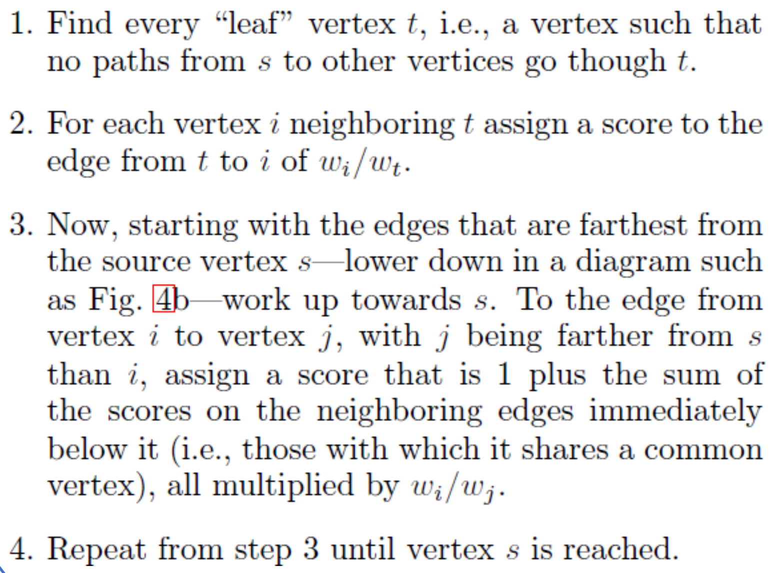


1. Find every "leaf" vertex t, i.e., a vertex such that no paths from s to other vertices go though t.
2. For each vertex i neighboring t assign a score to the edge from t to i of w_i/w_t.
3. Now, starting with the edges that are farthest from the source vertex s—lower down in a diagram such as Fig. 4b—work up towards s. To the edge from vertex i to vertex j, with j being farther from s than i, assign a score that is 1 plus the sum of the scores on the neighboring edges immediately below it (i.e., those with which it shares a common vertex), all multiplied by w_i/w_j.
4. Repeat from step 3 until vertex s is reached.

Calculation of shortest-path betweenness:

Calculate the contribution to edge betweenness from all shortest paths starting at s.



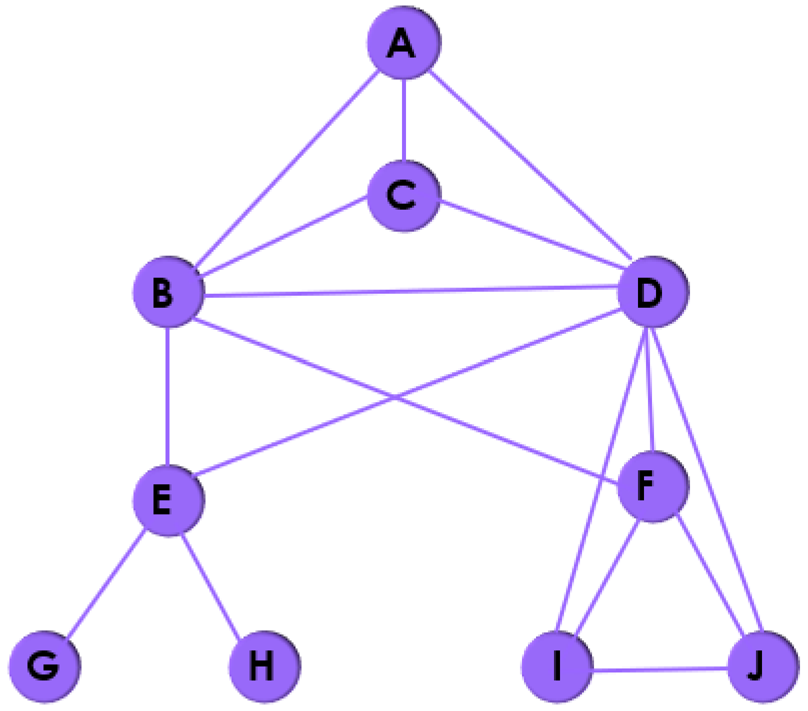


1. Find every "leaf" vertex t, i.e., a vertex such that no paths from s to other vertices go though t.
2. For each vertex i neighboring t assign a score to the edge from t to i of w_i/w_t.
3. Now, starting with the edges that are farthest from the source vertex s—lower down in a diagram such as Fig. 4b—work up towards s. To the edge from vertex i to vertex j, with j being farther from s than i, assign a score that is 1 plus the sum of the scores on the neighboring edges immediately below it (i.e., those with which it shares a common vertex), all multiplied by w_i/w_j.
4. Repeat from step 3 until vertex s is reached.

Community Detection: Girvan–Newman

Calculation of shortest-path betweenness:

Calculate the contribution to edge betweenness from all shortest paths starting at A.

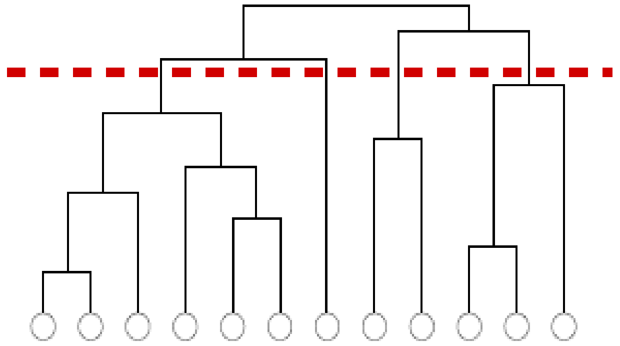


Community Detection: Girvan–Newman

Demo using NetworkX and Gephi

1. Girvan_Newman1.py
2. Girvan_Newman2.py
3. Girvan_Newman.py

Hierarchical clustering



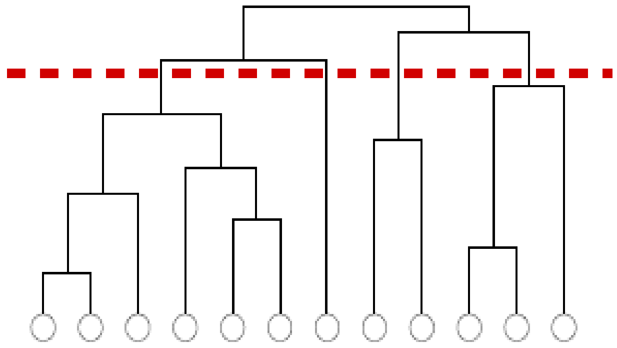
- The starting point of any hierarchical clustering method is the definition of a **similarity measure** between vertices.
  After a measure (Chebyshev distance, Minkowski distance …) is chosen, one computes the similarity for each pair of vertices, no matter if they are connected or not.
  At the end of this process, one is left with a new n×n matrix X, the similarity matrix.

- Hierarchical clustering techniques **aim at identifying groups of vertices with high similarity**, and can be classified in two categories:
  1. **Agglomerative algorithms**, in which clusters are iteratively merged if their similarity is sufficiently high;
  2. **Divisive algorithms**, in which clusters are iteratively split by removing edges connecting vertices with low similarity. (Eg. Girvan Newman)

Community Detection:

Hierarchical Clustering

- A **hierarchical tree** or **dendrogram** illustrating the type of output generated by the algorithms described here.
  The circles at the bottom of the figure represent the individual vertices of the network.
  As we move up the tree the vertices join together to form larger and larger communities, as indicated by the lines, until we reach the top, where all are joined together in a single community.
- Alternatively, the dendrogram depicts an initially connected network splitting into smaller and smaller communities as we go from top to bottom.
- A cross-section of the tree at any level, as indicated by the dotted line, will give the communities at that level.


Dendrogram

Note: Divisive and agglomerative methods grossly fail to identify the communities when overlaps are significant.

SOCIAL NETWORK ANALYTICS

Community Detection

Louvain Algorithm

Prakash C O

Department of Computer Science and Engineering

SOCIAL NETWORK ANALYTICS

Community Detection

Louvain Algorithm

Prakash C O

Department of Computer Science and Engineering

Community Detection

- The typical size of large networks such as social network services, mobile phone networks or the web now counts in millions when not billions of nodes and these **scales demand new methods to retrieve comprehensive information from their structure**.

- **A promising approach consists in decomposing the networks into sub-units or communities,** which are sets of highly inter-connected nodes.
- **The identification of these communities is of crucial importance** as they may help to uncover a-priori unknown functional modules such as topics in information networks or cyber-communities in social networks.

Community Detection

➢One can distinguish several types of community detection algorithms:

1. **Divisive algorithms** detect inter-community links and remove them from the network,
2. **Agglomerative algorithms** merge similar nodes/communities recursively and
3. **Modularity optimization methods** are based on the maximization of an objective function.

Community Detection

- **The exact modularity optimization is a problem that is computationally hard** and **so approximation algorithms are necessary when dealing with large networks**.

- The fastest approximation algorithm for optimizing modularity on large networks was proposed by Clauset et al.
  - Finding community structure in very large networks - Aaron Clauset, M. E. J. Newman, and Cristopher Moore

Community Detection

For a weighted graph, Clauset et al defined modularity Q as:

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

where

- A_{ij} represents the edge weight between nodes i and j;
- k_i and k_j are the sum of the weights of the edges attached to nodes i and j, respectively;
- m is the sum of all of the edge weights in the graph;
- c_i and c_j are the communities of the nodes; and
- δ is Kronecker delta function ($\delta(x, y) = 1$ if $x = y$, 0 otherwise).

Community Detection

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

- This greedy optimization method(Clauset et al) may produce values of modularity that are significantly lower than what can be found by using, for instance, simulated annealing.
- The Clauset et al method also has a tendency to produce super-communities that contain a large fraction of the nodes, even on synthetic networks that have no significant community structure.
- This method also has the disadvantage to slow down the algorithm considerably and makes it inapplicable to networks of more than a million nodes.
  This undesired effect of Clauset et. al method has been circumvented in Louvain method(Blondel et. al) by introducing tricks in order to balance the size of the communities being merged, thereby speeding up the running time and making it possible to deal with networks that have a few million nodes.

Community Detection

- The largest networks that have been dealt with so far in the literature are
  1. a protein-protein interaction network of 30739 nodes,
  2. a network of about 400000 items on sale on the website of a large on-line retailer, and
  3. a Japanese social networking systems of about 5.5 million users.

These sizes still leave considerable room for improvement considering that, as of today,

1. the social networking service Facebook has about 2.7 billion monthly active users,
2. the mobile network operator Vodafone has 302.6 million proportionate mobile customers across the globe and
3. Google indexes several billion web-pages.

In most large rea-world networks there are several natural organization levels– communities divide themselves into sub-communities– and it is thus desirable to obtain community detection methods that reveal this hierarchical structure.

Community Detection

- The Louvain method for community detection is a method to extract communities from large networks **created by Blondel *et al*. from the University of Louvain.**

- The inspiration for Louvain method of community detection is the **optimization of modularity** as the algorithm progresses.

- Modularity is a scale value between -1(non-modular clustering) and 1(fully modular clustering) that measures the density of edges inside communities to edges outside communities.

Community Detection: Louvain Algorithm

- **The Louvain method is a simple, efficient and easy-to-implement method for identifying communities in large networks.**

- The **Louvain** method has been used with success for networks of many different type and for sizes up to 100 million nodes and billions of links.

- **It is today one of the most widely used method for detecting communities in large networks.**

Community Detection: Louvain Algorithm

- The method is a greedy optimization method that attempts to optimize the "modularity" of a partition of the network.

- **The modularity optimization is performed in 2-steps.**
  - **First**, **the method looks for "small" communities by optimizing modularity locally.**
  - **Second**, **it aggregates nodes belonging to the same community and builds a new network whose nodes are the communities.**

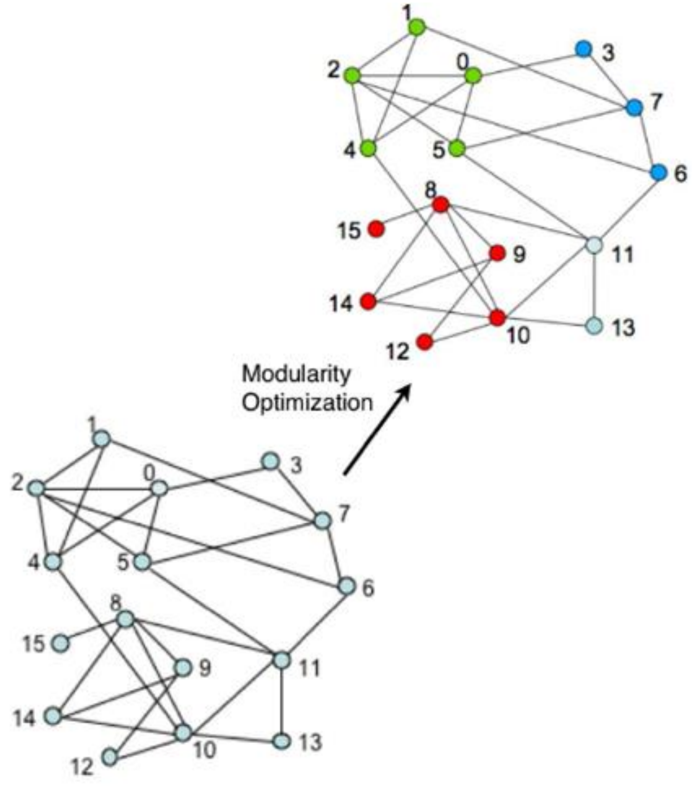
  These steps are repeated iteratively until a maximum of modularity is attained and a hierarchy of communities is produced.

- The exact computational complexity of the method is not known, the method seems to run in time O(n log n) with most of the computational effort spent on the optimization at the first level.

Community Detection: Louvain Algorithm

Phase-I (Louvain algorithm is divided in two phases that are repeated iteratively)

- Assume that we start with a weighted network of N nodes.
  - **First**, **assign a different community to each node of the network**. So, in this initial partition there are as many communities as there are nodes.
  - Then, **for each node i, consider the neighbors j of i and evaluate the gain of modularity that would take place by removing i from its community and by placing it in the community of j.**
  - **The node i is then placed in the community for which this gain is maximum**, but only if this gain is positive. If no positive gain is possible, i stays in its original community.
- **This process is applied repeatedly and sequentially for all nodes until no further improvement can be achieved and the first phase is then complete.**
- **This first phase stops when no individual move can improve the modularity**

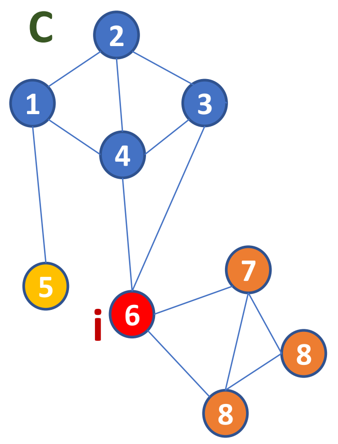


Community Detection: Louvain Algorithm

Part of the algorithm efficiency results from the fact that the gain in modularity ΔQ obtained by moving an isolated node i into a community C can easily be computed by:

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

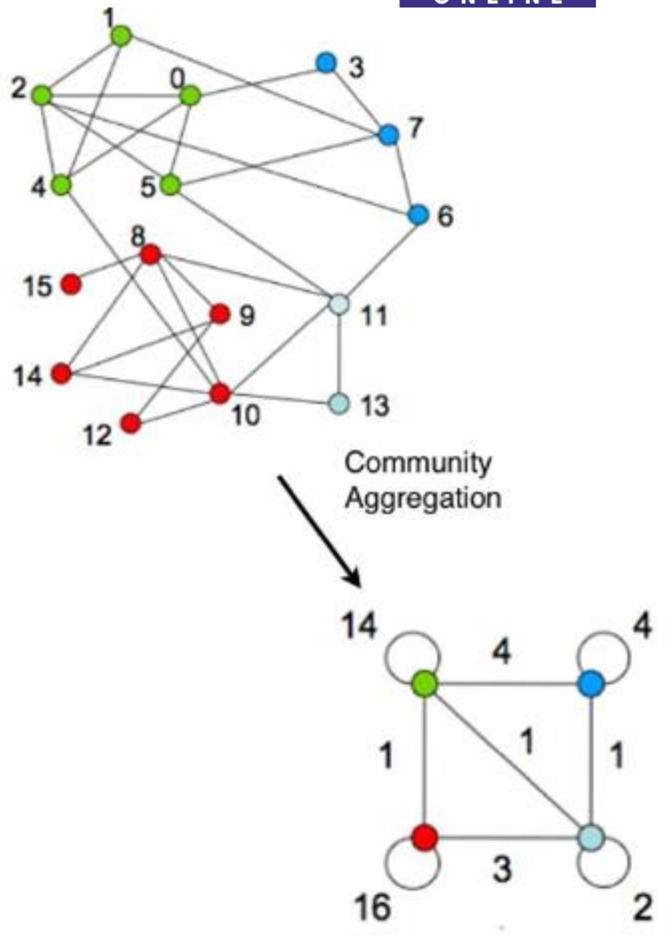
where $\sum_{in}$ is the sum of the weights of the links inside C, $\sum_{tot}$ is the sum of the weights of the links incident to nodes in C, k_i is the sum of the weights of the links incident to node i, $k_{i,in}$ is the sum of the weights of the links from i to nodes in C and m is the sum of the weights of all the links in the network. A similar expression is used in order to evaluate the change of modularity when i is removed from its community. In practice, one therefore evaluates the change of modularity by removing i from its community and then by moving it into a neighbouring community.



Community Detection: Louvain Algorithm

Phase-II (Community Aggregation)

- The second phase of the algorithm consists in building a new network whose nodes are now the communities found during the first phase.
- To do so, the weights of the links between the new nodes are given by the sum of the weight of the links between nodes in the corresponding two communities.
- Links between nodes of the same community lead to self-loops for this community in the new network.
- Once this second phase is completed, it is then possible to reapply the first phase of the algorithm to the resulting weighted network and to iterate.



Community Detection: Louvain Algorithm

- Let us denote by ”pass” a combination of these two phases.
- By construction, the number of meta-communities decreases at each pass, and as a consequence most of the computing time is used in the first pass.
- The passes are iterated (see Figure 1) until there are no more changes and a maximum of modularity is attained.

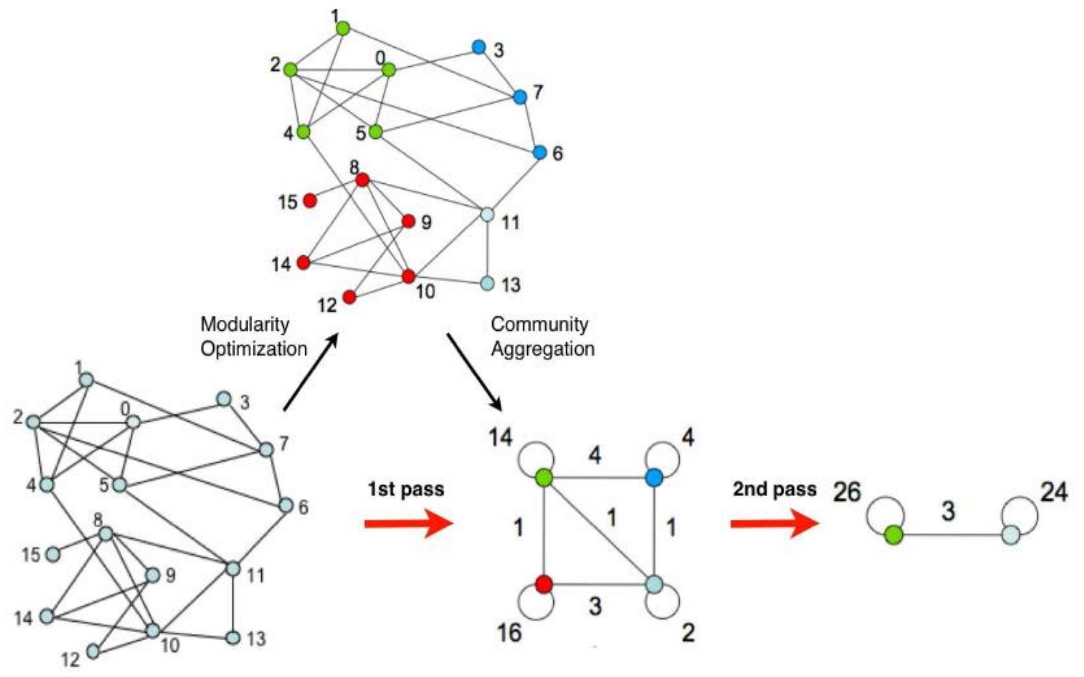


Figure 1. Visualization of the steps of our algorithm. Each pass is made of two phases:

Community Detection: Louvain Algorithm

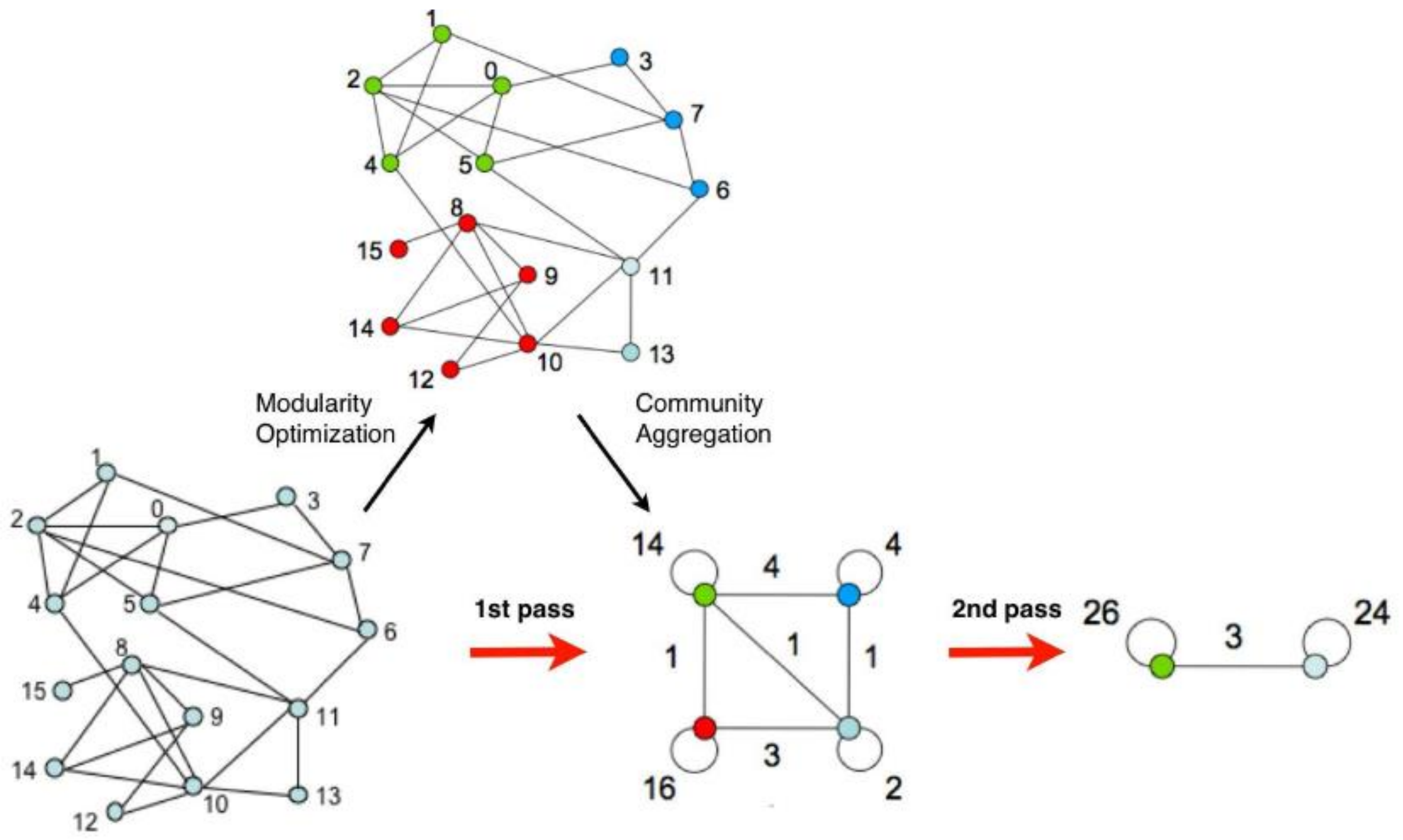


Figure 1. Visualization of the steps of our algorithm. Each pass is made of two phases:

Community Detection: Louvain Algorithm

➢Use-cases - when to use the Louvain algorithm

- **The Louvain method has been proposed to provide recommendations for Reddit users to find similar subreddits, based on the general user behavior**. Find more details, see "Subreddit Recommendations within Reddit Communities".
- **The Louvain method has been used to extract topics from online social platforms, such as Twitter and Youtube, based on the co-occurence graph of terms in documents, as a part of Topic Modeling process.** This process is described in "Topic Modeling based on Louvain method in Online Social Networks".
- **The Louvain method has been used to investigate the human brain, and find hierarchical community structures within the brain's functional network.** The study mentioned is "Hierarchical Modularity in Human Brain Functional Networks".

Community Detection: Louvain Algorithm

Conclusion

- The Louvain Modularity algorithm is used to evaluate social structures in Twitter, LinkedIn and YouTube.
- It’s also used in fraud analytics to evaluate whether a group has just a few bad behaviors or is acting as a fraud ring, which would be indicated by a higher relationship density than average.

Community Detection: Louvain Algorithm

Demo using NetworkX

1. CommunityDetection_Louvain.py

2. CommunityDetection.py

Demo using Gephi

References

- Finding and evaluating community structure in networks - M. E. J. Newman and M. Girvan
- Fast unfolding of communities in large networks - Vincent D. Blondel, Jean-Loup Guillaume, Renaud Lambiotte; and Etienne Lefebvre
- Community detection in graphs - Santo Fortunato
- Community detection in large-scale social networks – state of the art and future directions - Mehdi Azaouzi, Delel Rhouma, Lotf Ben Romdhane
- Wikipedia – Current Literature

THANK YOU

Prakash C O
Department of Computer Science and Engineering

coprakasha@pes.edu

+91 98 8059 1946

Community Detection

What is community structure?

- **Communities are often defined in terms of partition of the set of vertices, that is each node is put into one and only one community**. This is a useful simplification, and most community detection methods find this type of community structure.

- In some cases a better representation could be one where vertices are in more than one community.
  This might happen in a social network where each vertex represents a person, and **the communities represent the different groups of friends**: one community for family, another community for co-workers, one for friends in the same sports club, and so on.

Community Detection: Louvain Algorithm

- In order to maximize Modularity value efficiently, the Louvain Method has two phases that are repeated iteratively.
- First, each node in the network is assigned to its own community. Then for each node i, the change in modularity is calculated for removing i from its own community and moving it into the community of each neighbor j of i. This value is easily calculated by two steps:
  1. removing i from its original community, and
  2. inserting i to the community of j.

  The two equations are quite similar, and the equation for step (2) is:

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

Community Detection: Louvain Algorithm

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

Where Σ_{in} is sum of all the weights of the links inside the community i is moving into, Σ_{tot} is the sum of all the weights of the links to nodes in the community i is moving into, k_i is the weighted degree of i, $k_{i,in}$ is the sum of the weights of the links between i and other nodes in the community that i is moving into, and m is the sum of the weights of all links in the network.

Once modularity value is calculated for all communities i is connected to, i is placed into the community that resulted in the greatest modularity increase.

If no increase is possible, i remains in its original community.

This process is applied repeatedly and sequentially to all nodes until no modularity increase can occur. Once this local maximum of modularity is hit, the first phase has ended.

Community Detection: Louvain Algorithm

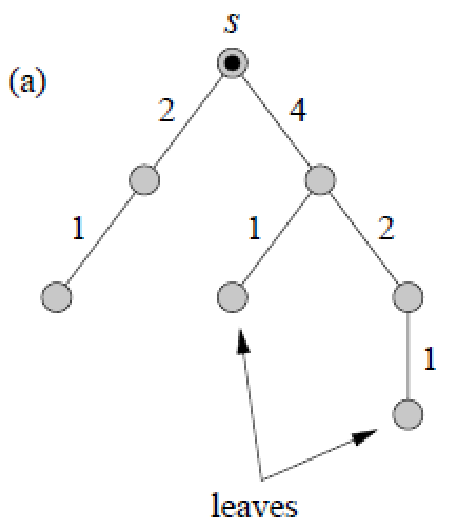
- In the Louvain Method of community detection, first small communities are found by optimizing modularity locally on all nodes, then each small community is grouped into one node and the first step is repeated.

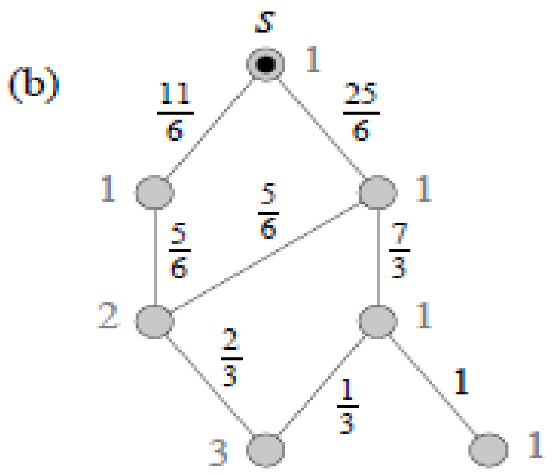
➢ The method is similar to the earlier method by Clauset, Newman and Moore that connects communities whose amalgamation produces the largest increase in modularity.

Community Detection: Girvan–Newman

Calculation of shortest-path betweenness:

In either case, we can check the results by confirming that the sum of the betweennesses of the edges connected to the source vertex is equal to the total number of reachable vertices—six in each of the cases illustrated here.


(a)
s
2
4
1
1
2
1
leaves


(b)
s
1
11/6
25/6
1
1
5/6
5/6
7/3
2
1
2/3
1/3
1
3
1