

SOCIAL NETWORK ANALYSIS

BAD714D

MODULE-4

Community Structure in Networks - Applications of Community Detection, Types of Communities, Community Detection Methods, Disjoint Community Detection, Overlapping Community Detection, Local Community Detection, Community Detection vs Community Search, Evaluation of Community Detection Methods.

T1 – Chapter 5 (5.1 – 5.8)

4.1 APPLICATIONS OF COMMUNITY DETECTION

This section presents various places where we can apply community detection algorithms.

1. **Link prediction:** Link prediction is used to detect future links in the network based on the current state of the network, i.e., current attributes of nodes and edges (see Chapter 6). Link prediction can also be used to find the missing links in the network. It has been observed that supplementing similarity-based link prediction algorithms with information about the community of a node helps in improving the performance of these algorithms.
2. **Recommender system:** The recommender system is used to find the preferences of users and recommend items accordingly (see Chapter 10). Community detection can be used to separate like-minded people, who might have similar tastes, and therefore might like similar kinds of products. For example, an e-commerce company does not have your past purchase history to suggest products that you might want to purchase. However, it has access to the underlying social network you are a part of, and knows which other people lie in your community. If it has purchase logs of other members of your community, it would be easier to suggest products to you, as both of you might have similar tastes.

3. **Information diffusion:** Information diffusion indicates how fast a particular piece of information or ideology spreads and what its overall reach will be (see Chapter 7). Generally, information spreads faster inside a community, and therefore, information about community membership can be used to maximise/minimise this spread. The more densely connected you are to other nodes inside a community, the more likely you are to share information with others or receive information from others. This may have an adverse effect when misinformation and fake news start to spread inside a community. To minimise the overall spread, one must first know the community membership of nodes so that we can take targeted actions. Political outfits might also leverage the underlying community structure of a network to analyse their influence and decide target communities for campaigning.
4. **Marketing:** Community detection is used to cluster users to do targeted marketing and better cater to customers' needs. They can also optimise the investments of a company by advertising only to some specific groups. Let us assume you want to advertise a product to a population. If you have a prior idea of a densely connected community in this population, there is a high probability that by introducing the product only to some community members, you can ensure that the entire community will soon know about it because of its closely knit structure. At the same time, you would also be able to save resources spent on advertising.
5. **Public health and epidemic spreading:** Community detection can be used to segregate various tissue in the case of diseases such as cancer and tumours. Community detection can also be used to find a group of people likely to be born with a particular deformity or a disease. The spread of an epidemic is also influenced by the network characteristics and their internal structure. A contagion that can spread through physical proximity will spread very quickly in dense communities. Therefore, an idea about the community structure can help us isolate and contain it more efficiently.
6. **Anomaly detection:** Anomaly is something that deviates from normal behaviour or expectation (see Chapter 8). Various anomaly detection algorithms use the community structure to predict expected behaviour and thus flag anomalous behaviour, especially in evolving networks. For instance, consider the previous example where we use community detection for improving recommendations of an e-commerce platform. It is possible that we might get some noisy inputs from

a few outlier customers, but we do not want such inputs to affect our recommendation systems. In such cases, we can still use the predictions made by the community detection pipeline to single out and remove outliers from consideration.

7. **Studying community evolution:** Dynamic community detection algorithms are used for predicting the evolution of a community. Using the current structure of a community and events, the task is to predict the future form of a community to check if it will expand, contract, merge, or dissolve. This kind of analysis can be useful for a host of applications. For example, it can help us analyse how a new transportation route affects the development of residential communities in an area.
8. **Criminology and detecting terrorist groups:** We can use a community detection algorithm to find criminal and terrorist groups. It can also be used to map the spread of criminal ideology and terrorist recruitment on social networks. If one has community information about the network, and has identified a few nodes to be malicious, then the entire community can be singled out and investigated, narrowing the search space for investigation authorities.

4.2 TYPES OF COMMUNITIES

Communities observed in large networks can be divided into four types based on their structural properties.

1. Disjoint communities: This type of community is also known as **flat communities**. Each node in the network can belong to at most one community, i.e., these communities do not overlap. An example of a flat community is that a person can be a full-time employee of only one organisation at a time. A toy example of a flat community is shown in Figure 5.1.

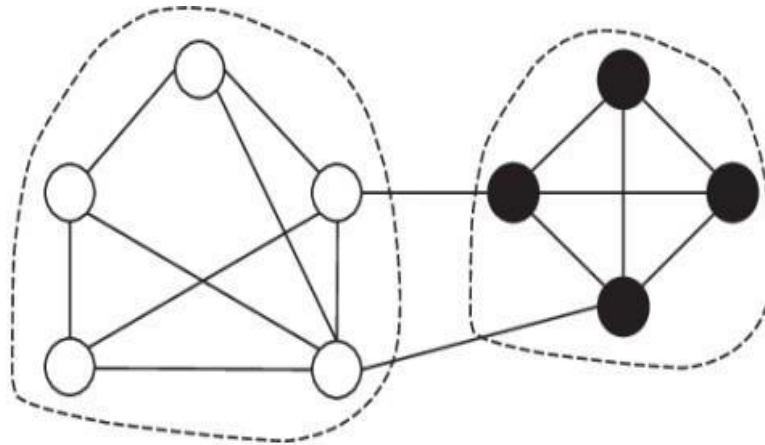


FIGURE 5.1. Disjoint communities.

It is worth noting that flat communities are different from disconnected components of a network. Nodes in two different communities can still have an edge between them, often referred to as a "**bridge**" between the communities. This is unlike disconnected components, where there are no edges between two nodes belonging to different components.

2. Overlapping communities: Members of these communities can belong to more than one community at a time. Two communities can even share an edge. An example of an overlapping community is a person belonging to multiple groups in a social network at a time. Overlapping community is the most realistic and generic community structure. Overlapping communities are harder to find than flat communities due to the possible exponential assignments of nodes into communities. A toy example of an overlapping community is shown in Figure 5.2.

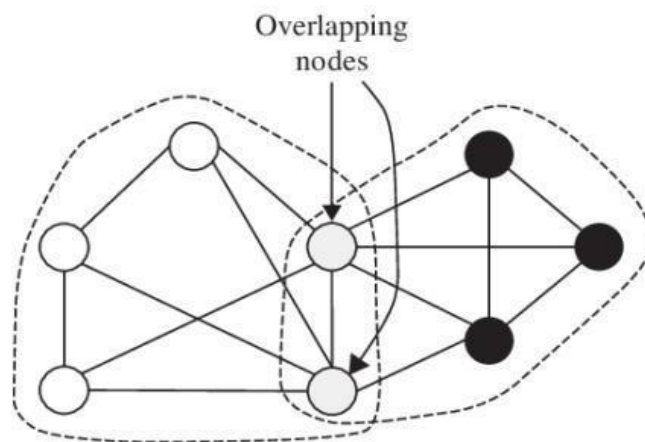


FIGURE 5.2. Overlapping communities.

3. Hierarchical communities: We can merge two or more flat or overlapping communities to form a hierarchical community. A hierarchical community can be linked to other hierarchical, overlapping, or flat communities. An example of the hierarchical community structure is that communities at the city level can be merged to form a community at the state level, which can further be merged to form country-level communities. A toy example of a hierarchical community is shown in Figure 5.3.

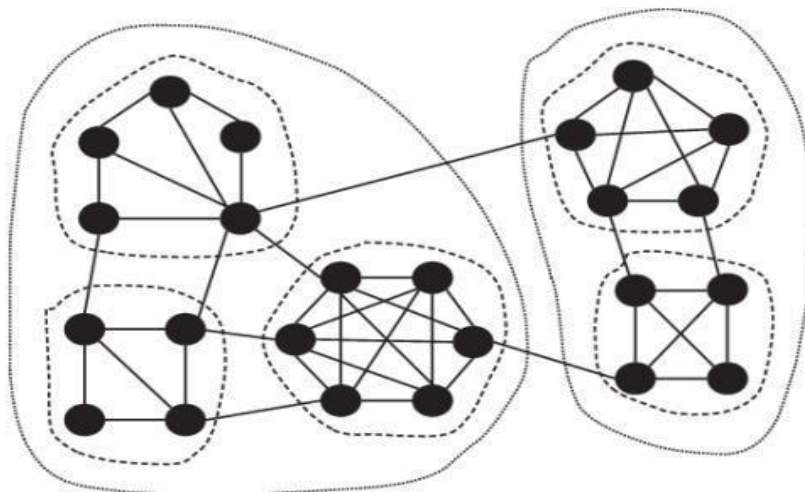


FIGURE 5.3. Hierarchical communities.

4. Local communities: Local communities show a community structure from a local perspective without focusing on the global structure. For example, consider the citation network formed by research groups inside a university. One may be able to find a local

community in this network. But when considering it as part of a more extensive global citation network, this community structure might not fit the requirements of a community anymore. A toy example of a local community is shown in Figure 5.4.

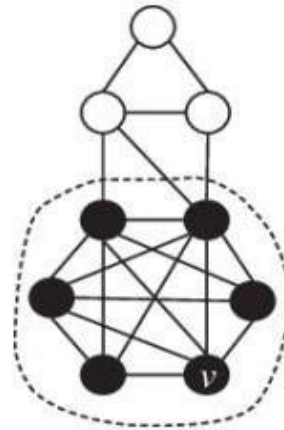


FIGURE 5.4. Local community of node v .

4.3 COMMUNITY DETECTION METHODS

In Section 5.2, we have discussed various types of communities present in the network. The critical question is to unfold the community structure of a network, particularly for cases where networks are gigantic. Community detection methods vary depending upon the nature of communities (disjoint, overlapping, or local) that we try to detect. A taxonomy of community detection methods is shown in Figure 5.5. All the ways of disjoint community detection will be discussed in detail in the next section, except for community detection in dynamic networks.

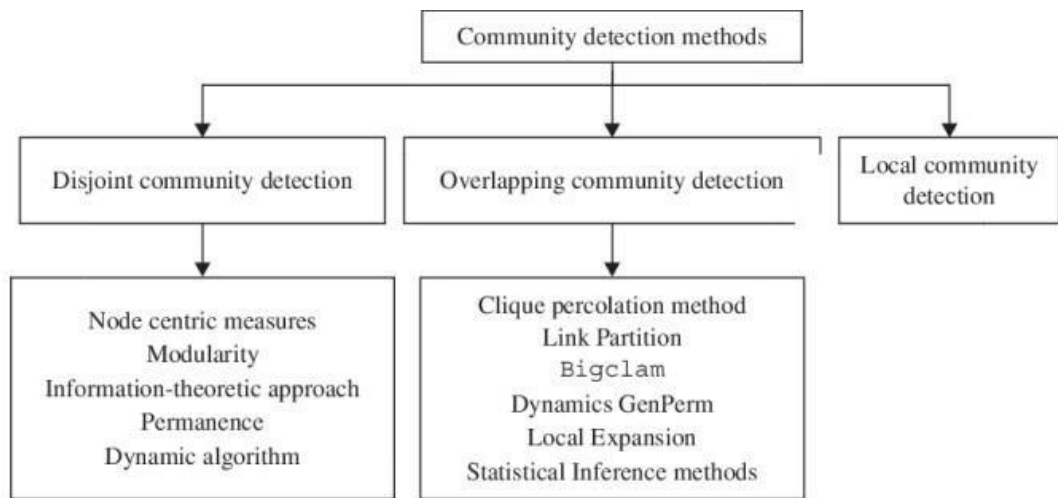


FIGURE 5.5. A taxonomy of community detection methods.

4.4 DISJOINT COMMUNITY DETECTION

This section presents various methods used to find disjoint communities in networks. These methods are based on features of nodes and edges in the network. Disjoint communities are easier to visualise, as a node can only belong to one community at a time.

4.4.1 Node-centric Community Detection

Node-centric community detection methods use the properties of nodes to find community structure in the network. This can be done in various ways, discussed in the subsequent sections.

Complete Mutuality

The complete mutuality of the network can be computed by finding the **cliques** in the network. A subgraph of a graph is a clique if every vertex-pair in the subgraph is adjacent, i.e., there is an edge between every vertex-pair of the subgraph. Cliques in networks can be considered as communities, as nodes in cliques are entirely connected. A clique is known as a **maximal clique** if it is not contained in any other clique. A clique has a diameter of 1.

Example 5.1

A toy example of a clique is shown in Figure 5.6, where there are two maximal cliques, one of size 5 and the other of size 3.

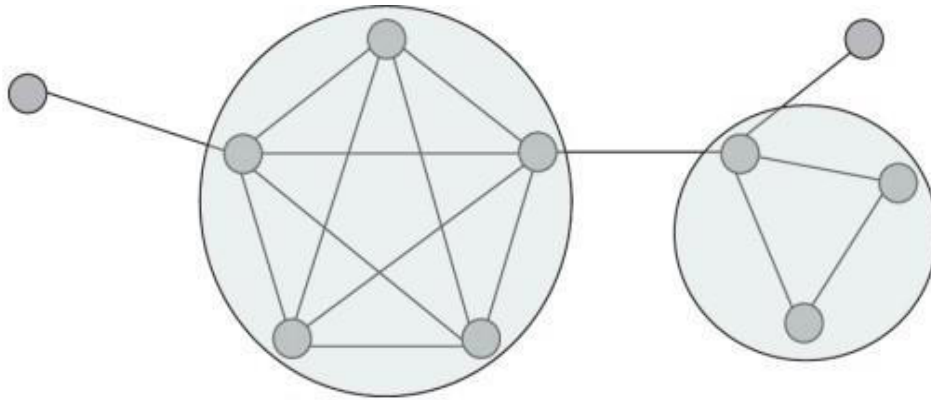


FIGURE 5.6. An example of a clique subgraph.

There are two problems with this approach:

1. Finding cliques from a network is **NP-complete**. Even though there exists an approximate algorithm to find the clique, this is still computationally expensive, as social networks are large.
2. Generally, large cliques are not present in social networks. There exist a few nodes that are not connected to all the other members in the subgraph. Consider a classroom group on Facebook, where most of the students are connected, but there are still some students who are not connected to all the other classmates.

Reachability of Members

As the constraints on cliques are too strict, this section presents a few measures in which the constraints are relaxed.

1. K-cliques: It is the maximal subset of vertices V such that for any two nodes belonging to this subset, the shortest distance between them is less than or equal to K . Here, a 1-clique is similar to a normal clique, as the distance between any two nodes is equal to one. A 2-clique is a maximal complete subgraph in which the distance between any two nodes is less than or equal to 2. In social network analysis, this is also known as a **friend of a friend**. The increasing value of K further relaxes the criterion of the clique.

Example 5.2

A toy example for a K -clique is shown in Figure 5.7, where nodes $\{1, 2, 3, 5, 6\}$ form a 2-clique.

The problem with K-clique is that despite a node not being present in a K-clique, it can contribute to finding the shortest distance. For example, consider Figure 5.7 again, where node 4 is not a part of the 2-clique, but it was still considered when finding the shortest distance between nodes 3 and 5.

2. K-clan: It is a stricter version of K-clique, in which only the nodes present in the set under inspection are used to create the subgraph, in which the distance between any two nodes should be less than or equal to **K**.

Figure 5.7 is not a 2-clan, but nodes {2, 3, 4, 5, 6} in Figure 5.8 form a 2-clan. Due to the maximality principle of K-clan, nodes 3, 4, 5 cannot be considered as a K-clan, as there exist two more nodes 2, 6 whose inclusion into the set still preserves the K distance principle of K-clan.

3. K-club: K-club is a K-clan with the relaxed maximality condition. Due to this, $\{2, 3, 4\}$, $\{3, 4, 5\}$, $\{4, 5, 6\}$, $\{5, 6, 2\}$, $\{6, 2, 3\}$ in Figure 5.8 can all also be considered as a 2-club.

As K-clan is a stricter version of K-club and K-clique, every K-clan is a K-club and K-clique. The problem with these methods is that for a large value of K, these algorithms are still computationally expensive, and deciding an appropriate value of K is a challenging task.

Node Degree

Another way to relax the constraints on the clique is by finding measures related to the degree of a node. These are divided into two major types:

1. K-plex: A subset of nodes of a network V forms a K-plex if every node in this network has a degree greater than or equal to $|V| - K$. In other words, every member is connected to at least $|V| - K$ other members. For a value of $K = 1$, K-plex is the same as a clique in the network, as every node in a clique has a degree of $|V| - 1$. For a low value of K (2 or 3), K-plex can find small cohesive subgroups in the network, and these subgroups follow the low diameter and high connectivity properties of a clique. K-plex directly solves the problem of K-cliques and K-clubs by limiting the number of non-neighbours inside the community.

Example 5.3

For the network given in Figure 5.9, the set of nodes forming a 4-plex is $\{1, 2, 3, 4, 5, 6\}$, and the set of nodes forming a 3-plex is $\{2, 4, 5, 6\}$.

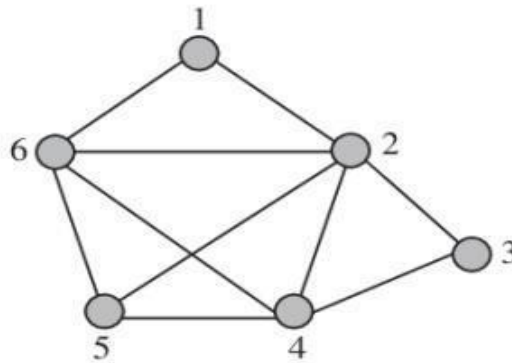


FIGURE 5.9. An example for a K -plex.

2. K-core: K-core is a maximal subgraph such that each node has a minimum degree of K . A $K + 1$ core subgraph can be created from the current subgraph by recursively removing nodes of degree K . This should be repeated until there is no node of degree K in the current subgraph. In the case of a social network, it is observed that a particular person leaves a group if less than a certain number of her friends are in that group.

Example 5.4

A toy example of a K-core is shown in Figure 5.10.

Example 5.4

A toy example of a K -core is shown in Figure 5.10.

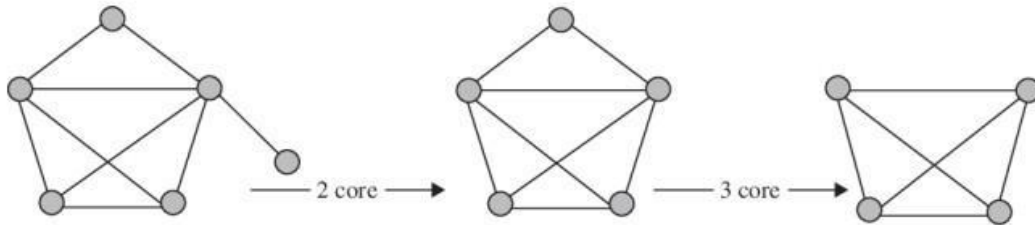


FIGURE 5.10. An example of a K -core.

Checking a given network for a K -core or K -plex is computationally easy, but finding the maximal K -core or K -plex is still an NP-complete problem.

4.4.2 Modularity and Community Detection

The node-centric methods discussed earlier are not helpful when the network is extensive. Therefore, we need new strategies for detecting communities in large networks. In this section, we discuss a method proposed by Newman (2006), known as **modularity maximisation**, which uses similarity and differences in network structure to find communities in the network.

Modularity

Modularity comes from the word "module," meaning a self-contained unit that can be used for making complex structures. Modularity is a network-centric metric to determine the quality of a community structure. It is based on the principle of comparison between the actual number of edges in a subgraph and its expected number of edges. The expected number of edges is calculated by assuming a **null model** in which each vertex is randomly connected to other vertices irrespective of the community structure, but keeping some of the structural properties the same (degree distribution).

For two given nodes i and j , the difference between the actual number of edges and the expected number of edges can be written as:

$$a_{ij} - \frac{\deg(i)\deg(j)}{2|E|} \quad (5.1)$$

where $\mathbf{A}$ is the adjacency matrix of network $G(V, E)$, and a_{ij} is the number of edges between nodes i and j in the network. $\deg(i)$ and $\deg(j)$ are the degrees of nodes i and j , respectively, and $|E|$ is the number of edges in $\mathbf{G}$, which is:

$$|E| = \frac{\sum_{v \in V} \deg(v)}{2} \quad (5.2)$$

The second term of Equation 5.1 is derived based on the concept of the **configuration model**.

Calculating the expected number of edges between nodes

Given a network $G(V, E)$, let us assume that a random network $G'(V, E')$ is created as follows: (a) the set of nodes in both $\mathbf{G}$ and G' remains the same; (b) the number of edges in both $\mathbf{G}$ and G' remains the same, i.e., $|E| = |E'|$; however, nodes in G' are connected through random rewiring (as discussed in the case of random networks in Chapter 3), resulting in a different set of edges; (c) the degree distribution of both networks is the same. Therefore, G' can be considered as the random counterpart of $\mathbf{G}$ (or the **null model** for $\mathbf{G}$).

Now consider two nodes v and w , with node degrees $\deg(v)$ and $\deg(w)$, respectively, from a randomly rewired network as described earlier. The expected number of edges between these nodes is $\frac{\deg(v)\deg(w)}{2|E|}$.

The modularity of the community structure can be written as:

$$Q = \frac{1}{2|E|} \sum_{i,j} \left(a_{ij} - \frac{\deg(i)\deg(j)}{2|E|} \right) \delta(\text{Comm}(i) \text{Comm}(j)) \quad (5.3)$$

where $\text{Comm}(i)$ is the community id of node i , $\delta(\text{Comm}(i) \text{Comm}(j))$ is 1 if vertices i and j belong to the same community (i.e., $\text{Comm}(i) = \text{Comm}(j)$), and 0 otherwise.

Modularity can also be written in terms of the number of edges as follows:

$$Q = \sum_{n=1}^{|\text{Comm}|} \left[\left(\frac{m_n}{|E|} \right) - \left(\frac{k_n}{2|E|} \right)^2 \right] \quad (5.4)$$

where m_n is the number of edges in community n , $|Comm|$ is the total number of communities, and k_n is the sum of the degree of nodes within community n .

Before we move ahead, let us understand what Equation 5.3 implies. In order for modularity to be maximised, what we want is to assign nodes into communities in such a way that every node is as densely connected to the nodes in its own community as possible. To understand when the value of this summation will be higher, let us first think about when an individual term in this summation will be higher. For the individual term in the summation to be higher, we require that both nodes have lower degrees (so that $\deg(i)\deg(j)$ is lower) relative to the total number of edges in the network. It means that we want an assignment where most of the actual edges of a node go within their own community.

The modularity of a network can be positive, negative, or zero, where a positive value shows the presence of a strong community structure. The value of modularity for unweighted and undirected networks lies in the range $[-1/21]$ (Brandes et al. 2007). An example of all three cases is given in Figure 5.11. Generally, we find the community structure such that the modularity value for the network is maximised.

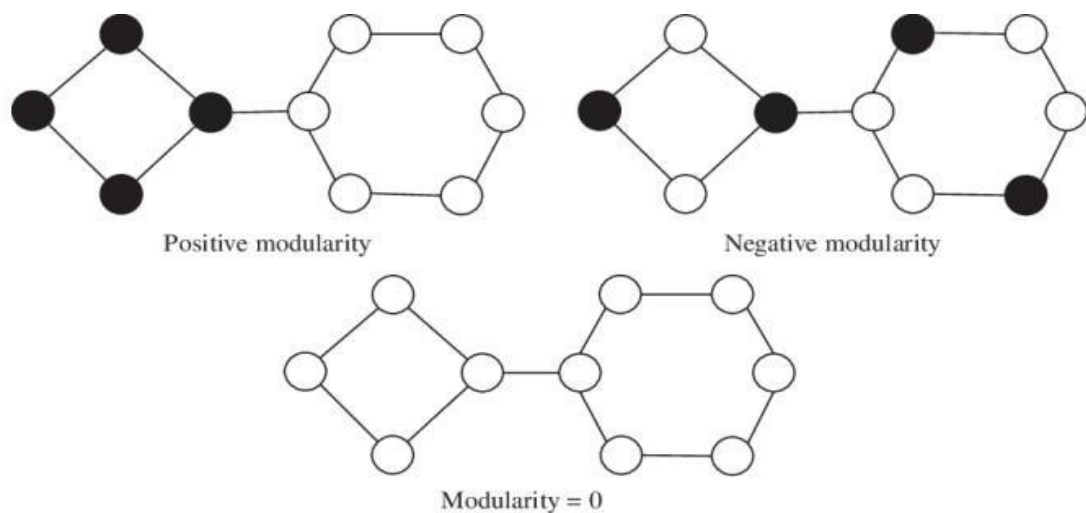


FIGURE 5.11. Different cases for modularity.

Example 5.5

For the network given in Figure 5.12, we first see how the difference between the actual number of edges and the expected number of edges is calculated, as shown in the attached table.

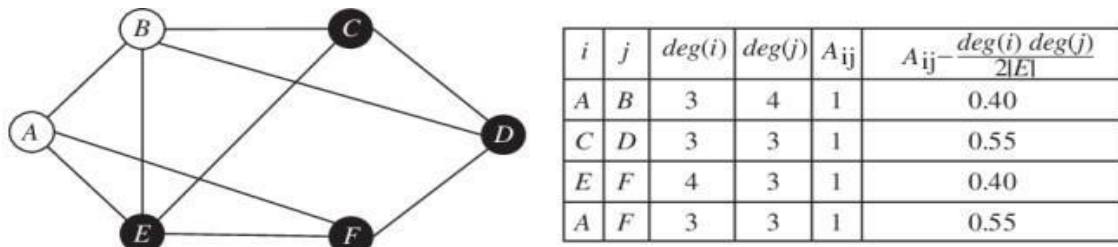


FIGURE 5.12. Difference between the actual and expected number of edges between some pairs of nodes.

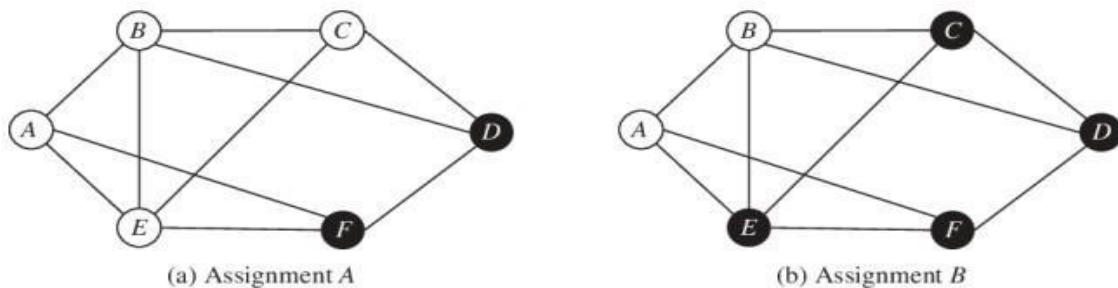


FIGURE 5.13. Different configurations of communities have different modularity values.

Now, we see how different configurations of community assignment can lead to a difference in the modularity values. In Figure 5.13, we have two community assignments of the same network. Let us compute the modularity values of the X (white nodes) and Y (black nodes) communities in both scenarios.

For assignment A: Using Equation 5.3, we obtain:

$$\begin{aligned}
 Q_X &= \frac{1}{2 \times 10} ((0.4)_{A-B} + (0.4)_{B-C} + (0.4)_{C-E} + (0.4)_{A-E} + (0.2)_{B-E} \\
 &+ (0.4)_{E-A} + (0.4)_{C-B} + (0.4)_{E-C} + (0.4)_{B-A} + (0.2)_{E-B} \\
 &+ (-0.45)_{A-C} + (-0.45)_{C-A} + (-0.45)_{A-A} + (-0.8)_{B-B} \\
 &+ (-0.45)_{C-C} + (-0.8)_{E-E}) \Rightarrow Q_X = 0.01
 \end{aligned}$$

$$\begin{aligned}
 Q_Y &= \frac{1}{2 \times 10} ((0.55)_{D-F} + (0.55)_{F-D} + (-0.45)_{F-F} + (-0.45)_{D-D}) \Rightarrow Q_Y \\
 &= 0.01
 \end{aligned}$$

For assignment B: Using Equation 5.3, we obtain:

$$Q_X = \frac{1}{2 \times 10} ((0.4)_{A-B} + (0.4)_{B-A} + (-0.45)_{A-A} + (-0.8)_{B-B}) \Rightarrow Q_X = -0.0225$$

$$Q_Y = \frac{1}{2 \times 10} ((0.55)_{C-D} + (0.55)_{D-F} + (0.4)_{C-E} + (0.4)_{E-F} + (-0.45)_{C-C} + (0.55)_{D-C} + (0.55)_{F-D} + (0.4)_{E-C} + (0.4)_{F-E} + (-0.45)_{C-F} + (-0.45)_{F-C} + (-0.45)_{D-D} + (-0.6)_{D-E} + (-0.6)_{E-D} + (-0.8)_{E-E} + (-0.45)_{F-F}) \Rightarrow Q_Y = -0.0225$$

In these two setups, we observe how different community assignments can lead to different modularity values for each community. Ideally, we would prefer an assignment that maximises the modularity of the overall network, which in this case would be **Assignment A**.

Fast Greedy Algorithm

This is the first algorithm for modularity maximisation proposed by Clauset et al. (2004). This algorithm is based on the **agglomerative hierarchical clustering** method. The basic principle of the algorithm is to merge the vertices in the subgraph if the modularity value increases for the subgraph. The steps of the algorithm are as follows:

1. Make each vertex a separate community; therefore, we have a total of $n = |V|$ communities in the initial state.
2. Now add the first edge in the network with the maximum modularity value of that partition, and reduce the set from n to $n - 1$.
3. Now add the other edges using the step 2 principle, to that community whose modularity value increases.
4. If the added edge does not increase the modularity value of the partition, it means the edge is an internal edge of the existing community.
5. In the end, we have a number of communities ranging from 1 to n . Figure 5.14 demonstrates the first two iterations of the Fast Greedy algorithm for a given input network. In each iteration, we look at all the possible pairs of communities that can be merged. We find the two communities whose merging results in a

maximum increase in modularity. The merge which results in the highest increase is fixed, and we repeat the whole process.

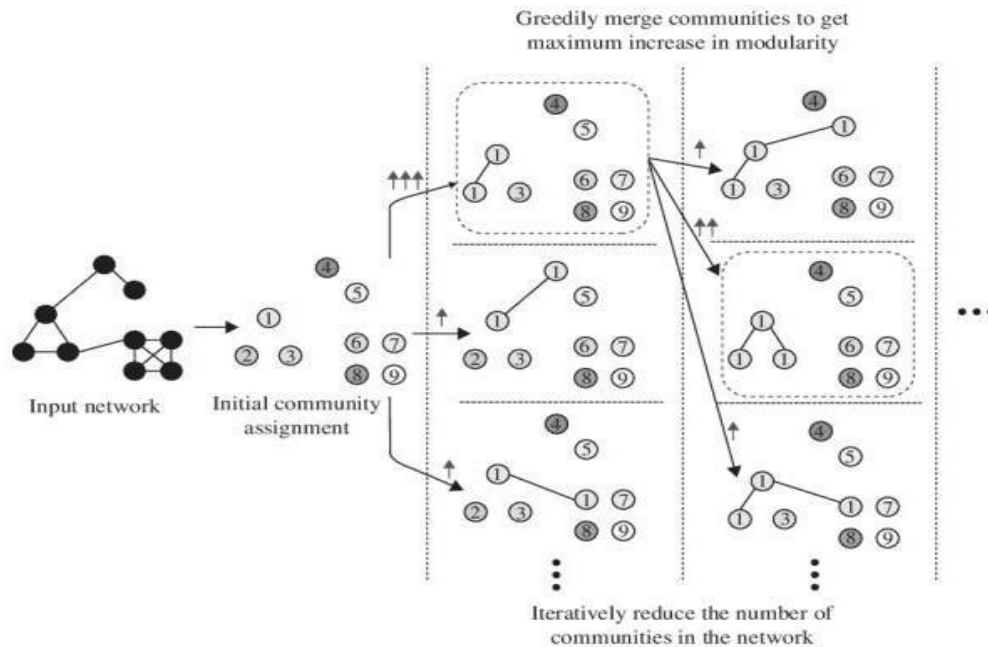


FIGURE 5.14. Toy example describing initial steps of Fast Greedy Algorithm. Initially, all the nodes belong to singleton communities. Then the communities are greedily merged to get a maximum increase in modularity.

The maximum modularity value of all the partitions in the network is the maximum approximation of the modularity value obtained from the algorithm. To determine the best merger, one should calculate the difference between previous and current modularity values and add to that cluster with the maximum difference.

The time complexity is $O(|E|)$ for joining edges in the network, where $|E|$ is the number of edges in the network. The total time complexity is $O((m+n)n)$, which includes the updation of the adjacency matrix.

Louvain Method for Community Detection

Louvain was introduced by Blondel et al. (2008) to approximate modularity maximisation in a greedy manner. The Louvain algorithm entirely unfolds the hierarchical structure of the community. The algorithm is divided into two stages in each iteration. It assumes the weighted network to identify communities. The stages are described as follows and visualised in Figure 5.15.

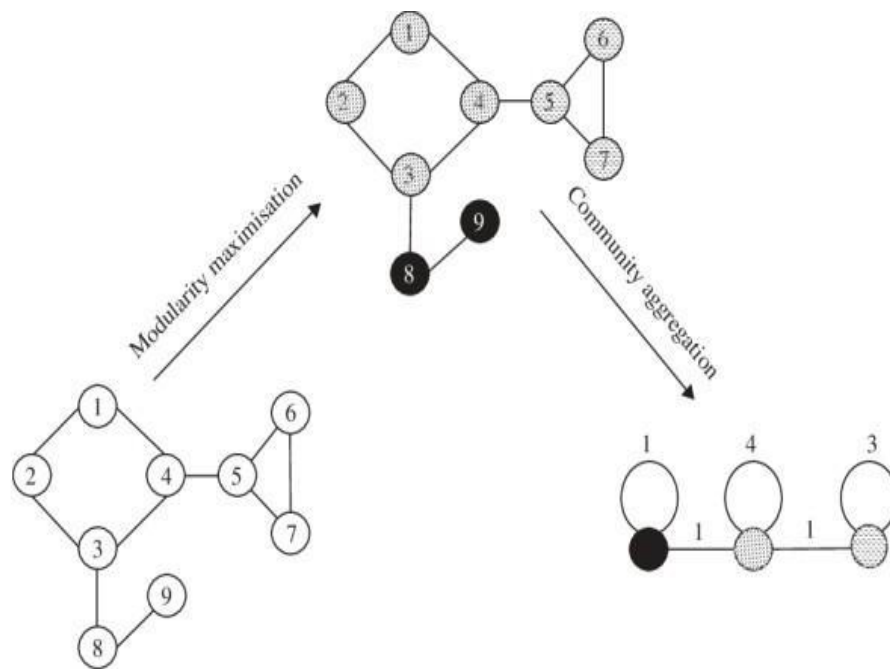


FIGURE 5.15. Two stages of the Louvain algorithm per iteration. Different shades of nodes indicate different communities.

1. First stage: Modularity maximisation

The following steps are used in the first stage of the algorithm.

1. Assign each node to a singleton community; therefore, we have a total of $n = |V|$ communities in the initial state. Select any one node as a seed node (multiple nodes can also be chosen).
2. Now add each node u to node v 's community if the change in the modularity value increases with this assignment.
3. Similarly, keep adding each node to that community which, upon addition, has maximum modularity gain.
4. Merging operation stops when there is no gain in the modularity value.

The output of this stage would be a node-community assignment.

2. Second stage: Node aggregation

In this stage, first we build the super-network whose nodes are the communities that are available in the first stage. The following steps are used for making a super-network and finding final communities.

1. For making a super-network, the edges between nodes are defined as the sum of weights of edges between the two communities.
2. The links between the nodes of the same community are aggregated as the self-loops of the nodes of the super-network, and the sum of their weights is the weight of the self-loop edge.

In the next iteration, repeat the two stages until the local maxima of the modularity value is reached. The nodes of the final super-network are the final communities of the given network. Selecting good seed nodes helps to converge the algorithm faster.

The time complexity of this algorithm is $O(n \log n)$, which is faster than the Fast Greedy algorithm.

Problems with Modularity Maximisation

Modularity maximisation suffers from a few problems. The main problem is the **resolution limit** (Fortunato and Barthélemy 2007; Kumpula et al. 2007) — well-connected smaller communities tend to merge with larger communities even if the resultant communities are not that dense. These small communities are well-defined, but loosely connected to the other part of the network. Modularity maximisation also fails to detect those well-separated communities with densely connected intra-community nodes but only a single inter-community edge with the rest of the network. It has been shown that even when two communities have just a single edge between them, as long as their sizes are below a certain threshold, they will be merged into a single community by modularity maximisation.

Another problem is the **degeneracy of solutions**. This refers to the case when there is an exponential number of community structures with the same (maximum) modularity value, resulting in different solutions being produced from the same network.

4.4.3 Information-theoretic Approaches

Information-theoretic approaches use compression techniques to estimate the best partition of a network. These approaches do not follow the idea of traditional cohesion and separation. Communities can be considered as some kind of regularities in the

network, and these approaches aim to use minimum information to represent the network. Thus, this requirement should ideally be less than the information required for the adjacency matrix.

Best compression is achieved by maximising the compactness, yet at the same time also minimising the loss. This will help in finding a simpler representation of the network, too. The quality of the representation is assessed using measures such as mutual information, minimum description length, etc. Based on the different representation and assessment measures, two popular algorithms of this kind are explained in the following section.

Infomod

Infomod, proposed by Rosvall and Bergstrom (2007), uses **mutual information** to find the best representation for network **X** using a candidate representation of network **Y**. As we are interested in finding the best community structure, we shall iterate over all the possible community assignments and select the community that gives maximum mutual information.

Let the initial representation of network **X** be the adjacency matrix of the network (or any other initial representation of a network). Our goal is to find a simpler description of **Y** such that the information captured is maximised. Therefore, we find $\hat{a}$, the community association for nodes such that the mutual information is maximised over all possible assignments of nodes.

$$\hat{a} = \arg \max_a I(X; Y) \quad (5.5)$$

where $I(X; Y) = H(X) - H(X | Y)$, $H(X)$ is the information necessary to describe **X**, and $H(X | Y)$ is the information necessary to describe **X** given **Y**. We, therefore, attempt to minimise $H(X | Y)$, which is defined as:

$$H(X | Y) = \log \left[\prod_{i=1}^{|\text{Comm}|} \binom{n_i(n_i - 1)/2}{m_{ii}} \prod_{i>j} \binom{n_i n_j}{m_{ij}} \right] \quad (5.6)$$

where the notations within parentheses are **binomial coefficients**, n_i is the size of community i , $|Comm|$ is the number of communities, m_{ii} is the number of edges within i , and m_{ij} is the number of edges between communities i and j .

As it is impossible to iterate over all the possible combinations of communities, **simulated annealing** is used to find the community structure that maximises the mutual information. In addition to this, having prior information regarding the number of communities is not always possible. Thus, Infomod uses **MDL (minimum description length)** to find the appropriate number of communities along with the description Y for the communities.

Infomap

Infomap is another approach proposed by Rosvall et al. (2009), which is used to detect community structure specifically in weighted and directed networks. This approach is based on the idea that a good map presents a good deal of information and requires a minimum bandwidth.

Here, random walks are done to measure the flow of information in the network. Therefore, the group of nodes in which the information flows faster in comparison can be aggregated to form a single community.

Using the two-level nomenclature of **Huffman coding**, efficient codes are generated to first describe nodes in a community (first-level) and to describe communities in the network (second-level). This two-level architecture allows us to describe information using the minimum amount of data.

Ideally, as the partitions would contain fewer inter-community links, random walkers will spend more time with the links themselves. Therefore, this would enable us to use the second-level information to describe the path, and additionally, other node-related information can be captured from the first-level description. This can be combined with MDL and simulated annealing to find appropriate encoding for the nodes.

Problems with Information-theoretic Approaches

Information-theoretic approaches for community detection are not flawless. The following are some of the limitations of the approaches in this category:

1. Simulated annealing used in information-theoretic approaches suffers from poor searching of global maxima.

2. The efficiency of clustering decreases when the structure is not apparent.
3. These approaches can lead to the formation of certain outlier communities.
4. These approaches also suffer from the resolution limit problem.
5. Such methods exhibit a "view" problem, which establishes an upper limit on the size of the community.

4.4.4 Permanence and Community Detection

As we have discussed previously, modularity maximisation for community detection suffers from problems such as resolution limit and degeneracy of solutions. Modularity as a metric also focuses only on the connections of the node within its community. The impact of the external connections of a node is not explicitly taken into account anywhere. Modularity is a network-centric global metric that, when used for maximisation, considers the entire network structure. Thus, for a large and evolving network such as the Facebook friendship network, it is often difficult to run the modularity maximisation algorithm from scratch. Therefore, we need a method that looks at the local neighbourhood while detecting communities. In other words, we seek to design a local metric which, while optimised, does not need the entire network structure, but those portions of the network which have been modified recently. Such methods would be helpful for community detection in larger evolving networks.

With this motivation, Chakraborty et al. (2014) proposed a new metric, named **permanence**, which is a local metric for community detection. It turns out to be highly efficient while optimised for community detection (Chakraborty et al. 2016).

Permanence

Permanence is a **vertex-centric** metric. It measures up to what extent the neighbouring communities "pull" a given vertex. It is defined not only on the total number of internal connections of a node but also on the maximum number of external connections of a node to any single neighbouring community. If vertex **u** is in community **A** and vertex **v** is in community **B**, and if there exists an edge between **u** and **v**, then **A** and **B** are neighbouring communities of each other.

The underlying hypotheses of permanence are as follows:

Hypothesis 1: The number of internal connections of node **v** should be greater than the number of external connections of node **v** with any external community.

Hypothesis 2: In a community, all the vertices should be highly inter-connected to each other.

Combining these two ideas, the formula of permanence for vertex v is given as:

$$\text{Perm}(v) = \left[\frac{I(v)}{E_{\max}(v)} \times \frac{1}{\deg(v)} \right] - [1 - c_{in}(v)] \quad (5.7)$$

where $I(v)$ is the number of internal neighbours of v within its own community, $E_{\max}$ is the maximum number of connections of v to neighbours in an external community, and $\deg(v)$ is the degree of v . c_{in} is the internal clustering coefficient of v , which is defined by the ratio of the actual number of edges among the internal neighbours of v to the total possible edges among them. Figure 5.16 shows a toy example of permanence.

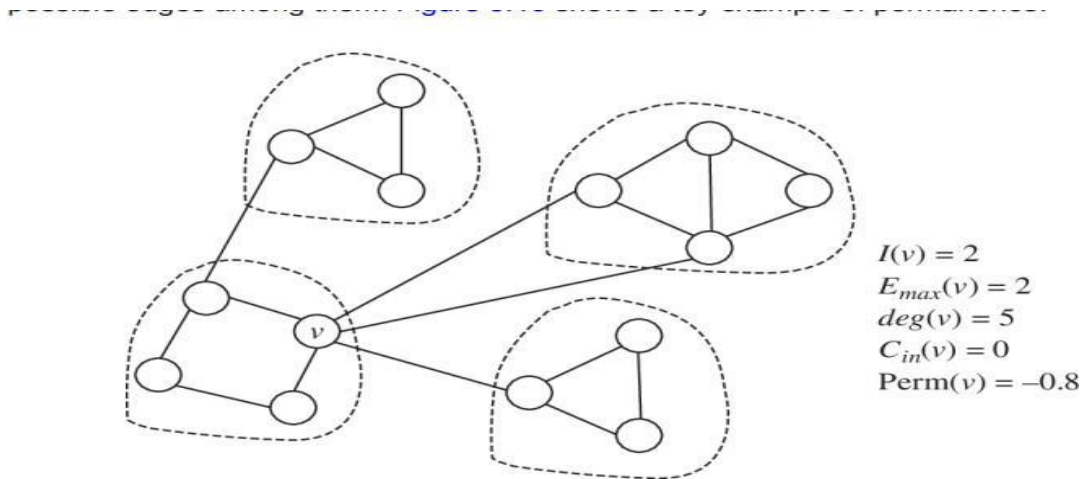


FIGURE 5.16. Calculating permanence of node v .

The permanence of the entire network is obtained by taking the average permanence over all the nodes:

$$\text{Perm}(G) = \frac{\sum_{v \in V} \text{Perm}(v)}{|V|} \quad (5.8)$$

The permanence value ranges between **-1 and 1**. A permanence of 1 can be obtained when vertex v is a part of a clique. Permanence will be 0 when there is no appropriate community structure of a network, such as a grid network (each node can be considered as a community, or the entire network is considered as a single community). Permanence

tends to be -1 when $I(v) \ll D(v)$, which means $\frac{I^{(v)}}{D(v)E_{max}(v)}$ is approximately 0 and $c_{in}(v)$ is 0. Chakraborty et al. (2014) mentioned that the permanence value of a network thus indicates if a network is eligible for community detection.

Example 5.6

Let us take the example discussed in Figure 5.13 again and see how community membership alters permanence scores for vertices C and E.

For Assignment A:

Vertex **deg(.)** **I(.)** **$E_{max}(.)$** **$c_{in}(.)$** **Perm(.)**

C 3 2 1 1 0.67

E 4 3 1 0.67 0.42

For Assignment B:

Vertex **D(.)** **I(.)** **$E_{max}(.)$** **$c_{in}(.)$** **Perm(.)**

C 3 2 1 0 -0.33

E 4 2 2 0 -0.75

Therefore, **Assignment A** will be preferred.

Permanence Maximisation for Community Detection

Chakraborty et al. (2014) further introduced **MaxPerm**, a permanence-maximisation based method for community detection. The algorithm uses a greedy approach for producing high-permanence partitions in the network. The basic principle of the algorithm is to join the small communities if and only if the permanence value of the network increases. The basic steps of this algorithm are the same as Louvain. MaxPerm is divided into two stages and assumes a weighted network to identify communities. The stages are described in detail as follows:

First stage: Permanence maximisation. The following steps are used in the first stage of the algorithm.

1. Assign each node to a single community; so for n number of nodes, there are n singleton communities. Select any node as a seed node (multiple seeds can be selected).
2. Add each node u to node v 's community if the change in permanence value increases.
3. Similarly, add each node to that community which has a maximum change in permanence value after adding the node.
4. Merging stops when the maximum permanence gain is attained.
5. Now the final network is developed with a number of communities ranging from 1 to n .

Second stage: Node aggregation. In this stage, first build the super-network whose nodes are the communities that are available in the final network of the first stage. The following steps are used for making a super-network and finding final communities.

1. For making a super-network, the edges between nodes are defined as the sum of weights of edges between two communities.
2. The links between the nodes of the same community are considered to form self-loops for the nodes of a super-network, and the sum of their weights is the weight of the self-loop.
3. Now apply the same steps of stage 1, and make the final network by combining nodes of a super-network only if there is a positive permanence gain.

The final nodes of the super-network generated by the maximum permanence algorithm are the final communities of the initial network.

Problems with Permanence Maximisation

Chakraborty et al. (2016) empirically and analytically showed that permanence maximisation reduces the resolution limit and degeneracy of solutions. However, if a vertex is connected to more than one neighbouring community, and those communities overlap with each other, then permanence fails to handle the small threshold value of the resolution limit. So again, the same problem arises that small communities are merged with giant communities and are not detected as they should be. For real-world networks, permanence maximisation tends to produce small communities.

Notation key for this section:

- Q = modularity score
- a_{ij} = actual edge indicator between nodes i, j (adjacency matrix entry)
- $\delta(\cdot, \cdot)$ = Kronecker delta (1 if same community, 0 otherwise)
- m_n, k_n = edges within community n , sum of degrees within community n
- $I(X; Y)$ = mutual information; $H(X), H(X | Y)$ = entropy, conditional entropy
- $\binom{n}{k}$ = binomial coefficient ("n choose k")
- $\text{Perm}(v)$ = permanence score of vertex v
- $I(v)$ = internal neighbour count; $E_{\max}(v)$ = max external connections to one community; $c_{\text{in}}(v)$ = internal clustering coefficient

5.5 OVERLAPPING COMMUNITY DETECTION**4.5.1 Clique Percolation**

Clique percolation, proposed by Palla et al. (2005), is the first method for overlapping community detection. It is based on the K -clique (here, K -clique is a complete subgraph of size k). First, we find all the K -cliques present in the network. Then, it attempts to merge two cliques if they have $K - 1$ nodes in common. This can be used to merge more than two cliques by fixing $K - 1$ nodes and then iteratively merging two cliques if they have these $K - 1$ nodes in common. The process stops when there are no more cliques left in the network.

With minor modifications, clique percolation can be applied to weighted and directed networks, known as **directed clique percolation (CPMd)** and **weighted clique percolation (CPMw)** methods, respectively.

Example 5.7

Consider the toy network given in Figure 5.17. Here, we take $K = 3$. The 3-cliques in the network are $\{1, 2, 4\}[A]$, $\{2, 3, 4\}[B]$, $\{3, 8, 9\}[C]$, $\{9, 5, 8\}[D]$, $\{5, 6, 7\}[E]$, $\{5, 6, 8\}[F]$. Using these, we create an intermediate network in which cliques are present as nodes, and there is an edge between cliques if they have $K - 1$ (here, 2) nodes from the original network in common.

The intermediate network for the above example is shown in Figure 5.18. As we can see, the intermediate network has two separate components. None of these components has a 3-clique. Therefore, the algorithm terminates. Hence, we obtain three communities as output, having nodes $\{0\}$, $\{1, 2, 3, 4\}$, $\{3, 5, 6, 7, 8, 9\}$, with node 3 as the overlapping node. This is shown in Figure 5.19.

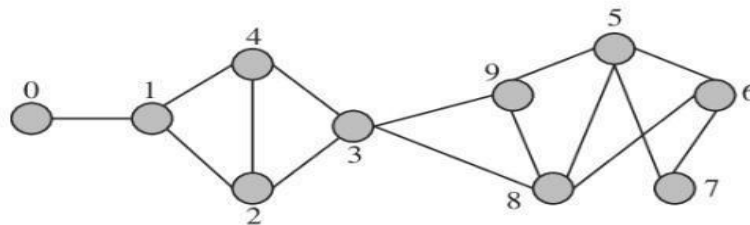


FIGURE 5.17. An example for clique percolation.

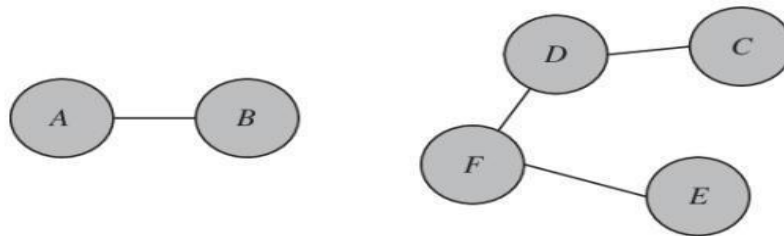


FIGURE 5.18. Intermediate step for clique percolation.

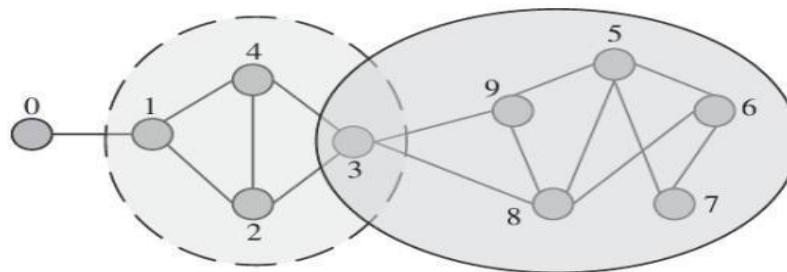


FIGURE 5.19. Final communities with clique percolation.

Problems with Clique Percolation Methods

1. There is no fixed value of K , and it is not easy to find a correct value of K . The value of K should not be too small or too large — if the value is too small, then the whole network becomes one big community; if it is too large, then it becomes hard to find the cliques.

2. Even though there are efficient implementations for the clique percolation method for networks with over a million nodes, finding a clique in a network is still computationally expensive.
3. This method is more like pattern matching applied to the network, as it tends to find localised structures in the network.

4.5.2 Link Partition

The link partitioning approach uses **links (edges)** in the network instead of nodes to detect communities (Ahn et al. 2010). If a node belongs to two or more communities (overlap), then links connected to it belong to two or more clusters. Similarity measures such as the **Jaccard coefficient** are used to cluster the edges.

For two edges e_{ik} and e_{jk} connected to node k , the value of the similarity metric, using the Jaccard coefficient, is given by:

$$\text{Sim}(e_{ik}, e_{jk}) = \frac{|N_i \cap N_j|}{|N_i \cup N_j|} \quad (5.9)$$

where N_i and N_j are neighbours of nodes i and j , respectively.

There are two major approaches for link partition:

1. Create a link network and apply a node-partitioning algorithm or disjoint community detection algorithm to find the community.
2. Use similarity measures on the edges to find the communities directly by creating the dendrogram.

Example 5.8

Find communities in the network shown in Figure 5.20 using link partition.

1. Using the link network approach, first create a link network where edges are the nodes. The link network corresponding to Figure 5.20 is shown in Figure 5.21. Let us assume that applying any disjoint community detection on this network gives us two communities: $\{\{1, 2\}, \{1, 4\}, \{3, 4\}, \{2, 4\}, \{2, 3\}\}$, and $\{\{5, 6\}, \{4, 5\}, \{4, 6\}\}$. Now, if we consider the vertices of the original network

covered by these communities, we obtain two communities: $\{1, 2, 3, 4\}$ and $\{4, 5, 6\}$.

- Using the link similarity-based approach, the answer is shown as a dendrogram in Figure 5.22.

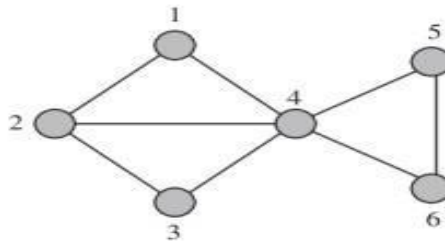


FIGURE 5.20. An example of link partition.

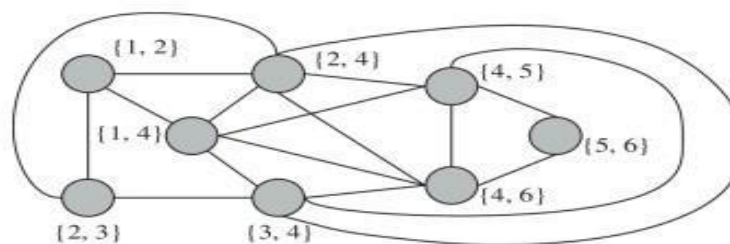


FIGURE 5.21. Link network corresponding to the network shown in Figure 5.20.

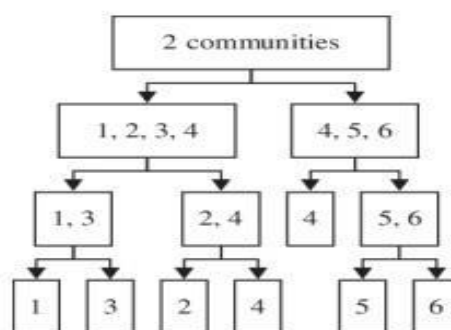


FIGURE 5.22. Link partition using similarity for the network shown in Figure 5.20. This is also called as 'dendrogram'.

The link-partitioning algorithm can also be applied to weighted and directed networks. The problem with link partition is that it is difficult to interpret the solution. In other

words, looking at the results, one might find it difficult to explain why the nodes have been assigned to the communities, as assigning links to communities does not make much sense.

4.5.3 BigClam

In the previous section, we saw algorithms that use network structure to generate a model. In contrast, the **BigClam** algorithm, proposed by Yang and Leskovec (2013), is based on a **generative modelling** approach. This means that we are given a network, and we attempt to find a model that best fits the network. This can be done in two steps:

1. Define models that can generate the required network.
2. Find a model that generates a network that best fits the required network.

The process of generative modelling thus, in turn, gives information about the community structure of the network. For this, we define the **community affiliation network** shown in Figure 5.23.

Figure 5.23.

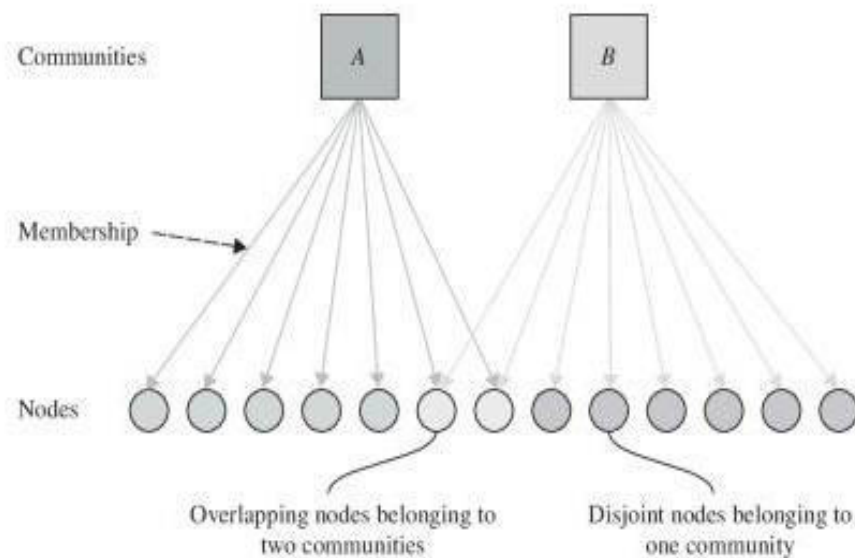


FIGURE 5.23. Community affiliation network for AGT.

For this, we define a generative model, aka the **affiliation graphical model (AGM)**, to generate a network G , which is parameterised as $G(V, C, M, P_c)$, where V is the nodes in the network, C is the set of communities, M is the association between nodes and communities, and P_c is the parameter associated with a particular community — it can also be considered as the probability of community c .

Therefore, we are given nodes in the final network, which are the same nodes as $\mathbf{V}$. What we need to determine is the edges in the network. Using AGM, we can find the probability of edges between two nodes as:

$$P(x, y) = 1 - \prod_{c \in M_x \cap M_y} (1 - P_c) \quad (5.10)$$

where M_x and M_y are the communities to which nodes x and y belong. The advantage of this kind of model is that it can be used to find disjoint as well as overlapping communities.

BigClam also uses a similar kind of graphical affiliation model, with the only difference being that each node and community membership is associated with a **strength** F_{xA} . We use the membership association of each node to define an **affiliation matrix** $\mathbf{F}$, where each row represents a node, and each column represents a community. Hence, using this modified definition, we can define the probability of two edges in one community as:

$$P_A(x, y) = 1 - \exp(-F_{xA} \cdot F_{yA}) \quad (5.11)$$

Using Equation 5.11, we define the overall probability of an edge between two nodes over all the communities as:

$$P(x, y) = 1 - \prod_c (1 - P_c(x, y)) \quad (5.12)$$

$$P(x, y) = 1 - \exp\left(\sum_c (-F_{xc} \cdot F_{yc})\right) \quad (5.13)$$

$$P(x, y) = 1 - \exp(-F_x \cdot F_y^T) \quad (5.14)$$

Now, given this information, what is required next is to find values for matrix $\mathbf{F}$. This can be done by finding the matrix F^* , which maximises the probability of network $\mathbf{G}$. Mathematically:

$$F^* = \arg \max_F L(F) \quad (5.15)$$

where

$$L(F) = \sum_{(x,y) \in E} \mathbf{G} P(x,y) - \sum_{(x,y) \notin E} \mathbf{G} (1 - P(x,y)) \quad (5.16)$$

Maximising $L(F)$ seems hard to attempt directly. Hence, we take the logarithm of both sides to simplify the equation. Now, this equation can be re-written as:

$$\arg \max_F \sum_{(x,y) \in E} (1 - \exp(-F_x \cdot F_y^T)) - \sum_{(x,y) \notin E} F_x \cdot F_y^T \quad (5.17)$$

This is a relatively simple problem that can be solved just by applying **gradient descent** over $L(F)$ to find values for matrix $\mathbf{F}$. For a single node $\mathbf{x}$, we write the equation as:

$$L(F_x) = \sum_{y \in n(x)} (1 - \exp(-F_x \cdot F_y^T)) - \sum_{y \notin n(x)} F_x \cdot F_y^T \quad (5.18)$$

For Equation 5.18, the gradient can be written as:

$$\nabla L(F_x) = \sum_{y \in n(x)} F_x \cdot \frac{\exp(-F_x \cdot F_y^T)}{1 - \exp(-F_x \cdot F_y^T)} - \sum_{y \notin n(x)} F_y \quad (5.19)$$

We update the value of row F_x as:

$$F_x = F_x + \eta \nabla L(F_x) \quad (5.20)$$

where η is the learning parameter. Even though taking the gradient is a linear-time algorithm, BigClam scales well with the size of the network and is fast compared to other community detection algorithms.

4.5.4 Dynamics

Most real-life networks are dynamic — they continuously evolve over time. Therefore, we require an algorithm that can directly be applied to the updated part only. One such semi-supervised machine learning algorithm is **label propagation**, which uses

information about previously labelled nodes to predict community labels for unlabelled ones. This algorithm is better than the previously proposed algorithms because it is more efficient and requires less information about the prior network structure.

General Algorithmic Steps

1. **Initialise** unique community labels for all the nodes in the network — each node is in its own community.
2. **Inner iteration:** Update labels for all the nodes in the network. A node's label is updated as the label with the highest frequency among its neighbours' current labels. In case of any discrepancy, randomly break the tie.
3. **Outer iteration:** If the label of the nodes in the previous iteration is the same as that for the node in the current iteration, then stop; otherwise, repeat the last step.

Label propagation is divided into two types based on the time at which the labels for nodes are updated:

1. Asynchronous mode: Labels are updated at the end of the inner iteration.

Example 5.9

An example of asynchronous mode is shown in Figure 5.24. Here:

1. Initial configuration, where nodes are labeled arbitrarily.
2. After the first inner iteration, update the label of node 2 by breaking the tie randomly and choosing node 1's label.
3. After the second inner iteration, update the label of node 3 by breaking the tie randomly and choosing node 4's label.
4. After the third inner iteration, update the label of a node by breaking the tie randomly and choosing node 3's label.
5. After the fourth inner iteration, update the label of node 5 by considering the majority labels of its neighbours.
6. After the fifth inner iteration, update the label of node 6 by considering the majority labels of its neighbours.

These steps are again repeated in the outer iteration, but labels will not change.

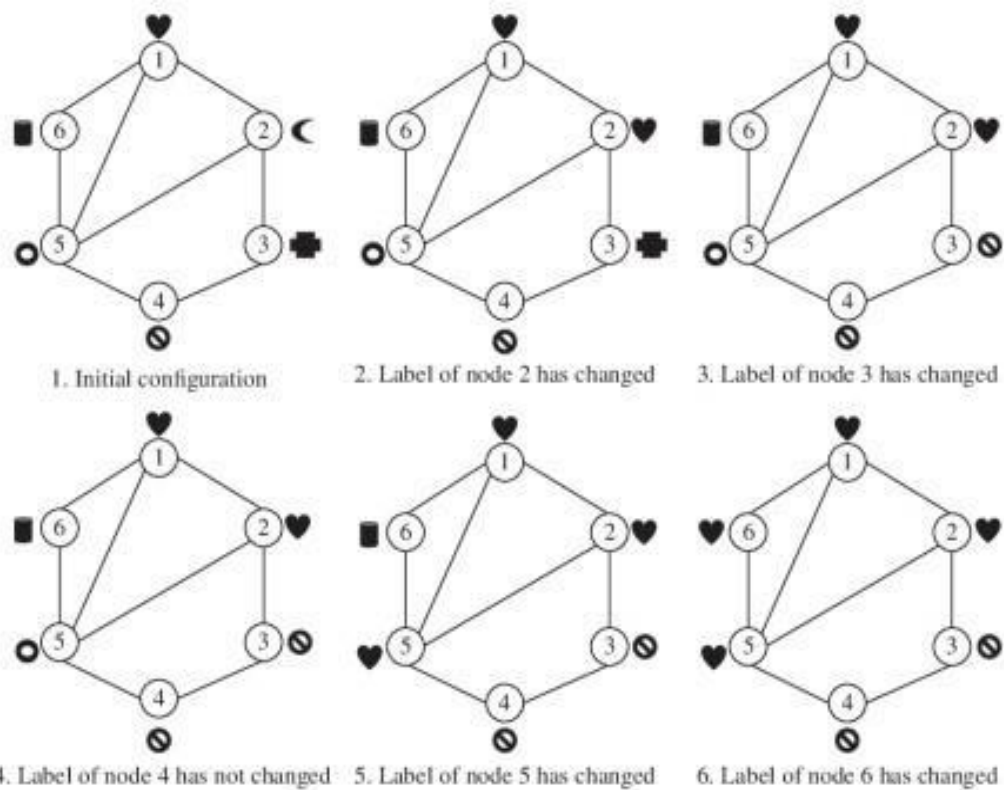


FIGURE 5.24. An example of asynchronous mode label propagation. Symbols associated with nodes are the labels (community ids).

2. Synchronous mode: Unlike asynchronous mode, within the inner iteration, labels of nodes are updated one by one sequentially; in this mode, labels are updated together at the end of the outer iteration.

Problems with Label Propagation

1. For large networks, it is not possible to find the number of outer iterations required to get the correct answer.
2. This method is not stable, as it involves a random process to break ties. Hence, you might end up with vastly different solutions in different runs, depending on the network structure.

3. It produces no unique solution, but a sequence of many solutions. However, the number of solutions can be narrowed down by getting a preliminary label for some nodes.

4.5.5 GenPerm: Generalised Permanence

GenPerm, an extension of permanence for both non-overlapping and overlapping communities, was proposed by Chakraborty et al. (2016). GenPerm provides us with information about the contribution of each node to the formation of every community. Although it is mainly developed for overlapping communities, it also works well (in fact, better than MaxPerm) for detecting non-overlapping communities. By design, GenPerm measures the contribution of each vertex in the network to every overlapping community that it is a part of.

Before we start with the actual algorithm, we need to familiarise ourselves with some basic terminologies. Let $\mathbf{G}$ be a network, and $\mathbf{u}$ and $\mathbf{v}$ be two vertices in the network.

1. **Shared edge:** The edge $\langle u, v \rangle$ is a shared edge if it lies in more than one community, i.e., $\exists i, j$ such that if C_i, C_j are communities, then $u, v \in C_i \wedge u, v \in C_j$. In Figure 5.26, edge $\langle v, C \rangle$ is a shared edge, as both v and C lie in the overlap of the two communities.

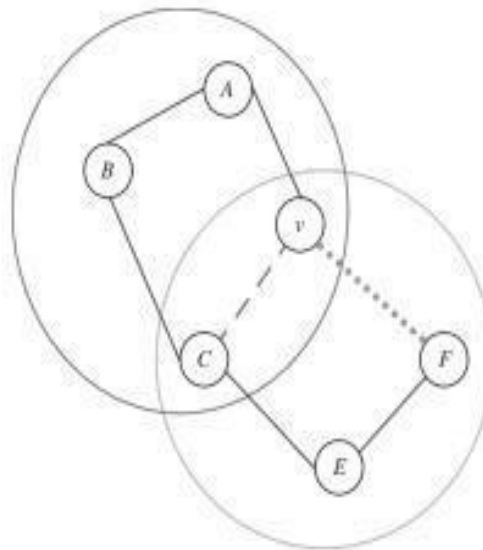


FIGURE 5.26. Illustration of shared edge $\langle v, C \rangle$ and non-shared edge $\langle v, F \rangle$ in a network.

2. Non-shared edge: The edge $\langle u, v \rangle$ is a non-shared edge if it lies entirely in a single community, i.e., $\exists! i, j$ such that if C_i, C_j are communities, then $u, v \in C_i \wedge u, v \in C_j$. In simple words, at least one edge of the vertices must lie in a single community only. In Figure 5.26, the edge $\langle v, F \rangle$ is a non-shared edge, as even though v lies in both communities, F lies in a single community.

GenPerm introduces two key concepts for each vertex $\mathbf{v}$ in community $\mathbf{c}$.

1. Generalising pull: One can think of generalising pull as the internal pull from its community. It is the sum of non-shared edges and the sum of shared borders, normalised by the total number of communities in which they are shared. The normalisation is required so that each edge only contributes an equal amount of pull to every community it is a part of.

$$I^c(v) = \sum_{e \in \Gamma_v^c} \frac{1}{x_e} \quad (5.21)$$

Here, Γ_v^c denotes the edges of vertex $\mathbf{v}$ that lie in community $\mathbf{c}$ (i.e., are connected to nodes in community $\mathbf{c}$), and x_e is the number of communities in $\mathbf{c}$ that contain edge $\mathbf{e}$.

2. Generalising connectedness: Generalising connectedness deals with how strongly vertex $\mathbf{v}$ is connected with the internal neighbours of the community.

$$(1 - c_{in}^c(v)) \times \frac{\sum_{e \in \Gamma_v^c} \frac{1}{x_e}}{I(v)} \quad (5.22)$$

where $c_{in}^c(v)$ is the local clustering coefficient of vertex $\mathbf{v}$, considering only the subgraph induced by community $\mathbf{c}$.

By using the above two concepts, GenPerm is defined as:

$$P_g^c(v) = \left(\frac{I^c(v)}{E_{max}(v)} \times \frac{1}{\deg(v)} \right) - (1 - c_{in}^c(v)) \times \frac{\sum_{e \in \Gamma_v^c} \frac{1}{x_e}}{I(v)} \quad (5.23)$$

Now, like MaxPerm, we assign nodes to communities so that GenPerm is maximised (we call it **MaxGenPerm**).

Example 5.11

We show how GenPerm is calculated for the network shown in Figure 5.27, where $\mathbf{v}$ is the centre node and a part of C_1 and C_3 communities, and C_2 is the neighbouring community.

Here, $\deg(v) = 8$ and $I(v) = 6$

$$P_g^{C1}(v) = \frac{1 + 1 + 1}{2 \times 8} - \left(1 - \frac{2}{3}\right) \times \frac{1 + 1 + 1}{6} = 0.02$$

$$P_g^{C3}(v) = \frac{1 + 1 + 1}{2 \times 8} - \left(1 - \frac{2}{3}\right) \times \frac{1 + 1 + 1}{6} = 0.02$$

Therefore, the total GenPerm value for $\mathbf{v}$ is $P_g(v) = 0.02 + 0.02 = 0.04$.

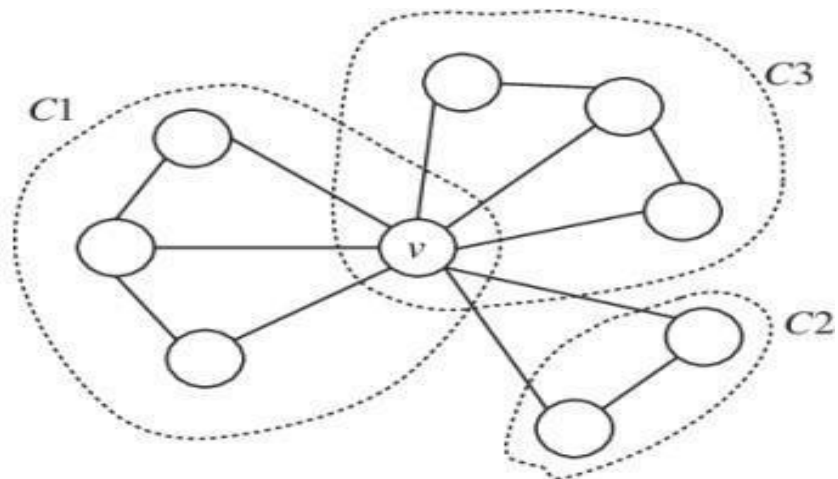


FIGURE 5.27. A toy network demonstrating GenPerm.

MaxGenPerm: Community Detection by Maximising GenPerm

Now that we have set up the basic terminology and the metric, let us look at the algorithm called **MaxGenPerm**. As previously stated, MaxGenPerm works in a manner similar to MaxPerm, except that this time, the metric being maximised is GenPerm, and the seed entities are edges (not nodes). The steps of the algorithm are described below:

1. **Initialisation:** First, initialise each edge of the network as a single community, so there is a chance that each edge belongs to multiple communities.
2. **Update:** In this step, first calculate the GenPerm value of vertex v with respect to each community it belongs to. If the value is greater than 0, then assign v to that community. This will develop a new set of communities for each vertex v . Now, calculate the total GenPerm value with respect to the new set of communities and check if it is greater than that in the previous step. If the value increases, then update the neighbouring community set of vertex v . Similarly, update the community set for all the vertices of the network.
3. **Convergence:** The algorithm stops updating when the GenPerm value does not increase for all the vertices, or the maximum number of iterations is attained.

Notation key for this section:

- $\text{Sim}(e_{ik}, e_{jk}) = \text{Jaccard similarity between two edges}$

- F = affiliation matrix (nodes $\times$ communities), with strength values F_{xA}
- $L(F)$ = likelihood function being maximised via gradient descent
- η = learning rate
- Γ_v^c = edges of vertex v lying in community c
- x_e = number of communities sharing edge e
- $c_{in}^c(v)$ = local clustering coefficient of v within community c
- $P_g^c(v)$ = GenPerm score of vertex v with respect to community c

4.6 LOCAL COMMUNITY DETECTION

4.6.1 Local Modularity

Local community detection comes into the picture when we have less knowledge about the global structure of the network, i.e., we know only a few edges around a given vertex. Therefore, to identify the community structure independent of the global properties, we rely on local community detection. Clauset (2005) proposed a metric named local modularity to identify the local community in the large network without knowing much about the global properties.

Consider an unweighted and undirected network G as shown in Figure 5.28. Let us say, we have complete knowledge about a subgraph C of G . The edges that go out of this subgraph will be connected to a set of vertices, U , any member of which does not lie in C but must have at least one neighbour in it. Through these vertices, we can explore the rest of G , and gain more knowledge about it. To do this, we pick a vertex $v \in U$, and include it in C . This way, we can include neighbour of v not already in C into U , and iteratively explore the whole network.

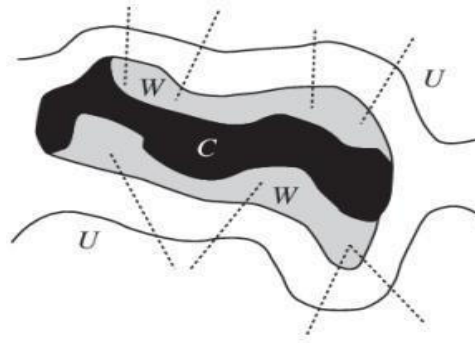


FIGURE 5.28. The abstract division of network G into local community C , its boundary W and the edges connecting W to the unknown neighbour U (Clauset 2005).

Now, let A denote the known adjacency matrix of G .

$$A_{ij} = \begin{cases} 1, & \text{if vertices } i \text{ and } j \text{ are connected, and } i \in C \vee j \in C \\ 0, & \text{otherwise} \end{cases}$$

If we consider C as a local community, one way that we can quantitatively define the quality of this community is by the fraction of known connections that lie completely inside of C . It can be formulated as,

$$\frac{\sum_{ij} A_{ij} \xi(i, j)}{\sum_{ij} A_{ij}} = \frac{1}{2m^*} \sum_{ij} A_{ij} \xi(i, j) \quad (5.24)$$

where $m^* = \frac{1}{2} \sum_{ij} A_{ij}$, i.e., the total number of edges in the partial adjacency matrix, and $\xi(i, j)$ is 1 when both i and j belong to C , and 0 otherwise.

Now, consider only those vertices in C , which have at least one neighbour in U , i.e., the “boundary” of the subgraph C . We denote this set of vertices by W . The boundary adjacency matrix $\hat{A}$ can be given by:

$$\hat{A}_{ij} = \begin{cases} 1, & \text{if vertices } i \text{ and } j \text{ are connected, and } i \in W \vee j \in W \\ 0, & \text{otherwise} \end{cases} \quad (5.25)$$

Using this boundary-matrix, local modularity Q_{local} is defined as,

$$Q_{\text{local}} = \frac{\sum_{ij} \hat{A}_{ij} \delta(i, j)}{\sum_{ij} \hat{A}_{ij}} = \frac{I}{T} \quad (5.26)$$

where $\delta(i, j)$ is 1, when i belongs to W and j belongs to C or vice versa, and 0 otherwise. T is the number of edges with at least one end-point in W , and I is the number of edges

having no end-point in U . The value of Q_{local} lies between 0 to 1. The reason for defining local modularity in this way is that the so-called boundary of community C is enough to give us an idea of how densely C is connected to the rest of the subgraph. A well-defined community will have fewer edges going from its boundary to the unknown network, as compared to the number of edges that lie wholly inside the community. Additionally, note how Q_{local} is independent of the specifications of the complete network G (unlike modularity), and it depends only on the specifications of the known local network, C .

Maximising Local Modularity

Based on our new notion of local modularity Q_{local} , we describe an algorithm proposed by Clauset (2005) to find local community. It takes polynomial time in terms of the number of vertices. The steps of the algorithm are as follows:

1. Let us start with a source vertex v_0 . Initially, the local community C consists only of v_0 . Consequently, the neighbouring vertices of v_0 will form U .
2. Iterating over all the neighbours of v_0 , we add that neighbours to C , which gives us the maximum increase in the value of Q_{local} .
3. Now that we have new vertices added to C , we explore their neighbours, and if we discover any previously unseen vertices, we add them to U .
4. This process continues until the number of vertices in C reaches a pre-decided maximum number.

The algorithm is based on an agglomerative greedy approach that may not give the best partition every time. While we can stop the algorithm after the number of vertices in C reaches a preset limit, we could also let the algorithm run until C grows to an arbitrarily large size. Doing this will give us a hierarchy of communities, each containing the source vertex v_0 .

4.6.2 Subgraph Modularity

While calculating local modularity, we only consider the boundary vertices of the subgraph and apply the greedy algorithm to find the local community in the network. But by fixing the size of the subgraph, we may not be able to find a highly optimised local community for every source vertex.

To overcome this issue, subgraph modularity is proposed based on the degree of each vertex. Now using the same network G , subgraph C , and the adjacent neighbours set U

that we defined for subgraph modularity, we define the adjacency matrix for subgraph C and its neighbours U as,

$$S_{ij} = \begin{cases} 1, & \text{if vertices } i \text{ and } j \text{ are connected, and } i \in C \vee j \in C \\ 0, & \text{otherwise} \end{cases} \quad (5.27)$$

Now, we define the in-degree and out-degree of subgraph C .

1. **In-degree:** The in-degree of subgraph C is the total number of edges that lie completely in subgraph C , that is, the number of edges, such that both their end-points lie in C :

$$\text{In}(C) = \sum_{ij} S_{ij} \delta(i, j), \quad (5.28)$$

where $\delta(i, j)$ is 1 if both i and j are in subgraph C , and 0 otherwise.

2. **Out-degree:** The out-degree of subgraph C is the total number of edges between C and the remaining part of the network G . Thus, out-degree is the number of edges such that exactly one of their end-points lies inside C .

$$\text{Out}(C) = \sum_{ij} S_{ij} \lambda(i, j),$$

where $\lambda(i, j)$ is 1 if exactly one of i or j lies in subgraph C , and 0 otherwise.

The subgraph modularity (SM) is defined for the subgraph C of network G as the ratio of the in-degree of subgraph C to the out-degree of the subgraph C .

$$\text{SM} = \frac{\text{In}(C)}{\text{Out}(C)}$$

Subgraph Modularity Maximisation Algorithm

The aim of the algorithm is to find the local community with maximum subgraph modularity that contains the source vertex. Let v be the source vertex, and the number of edges and vertices in subgraph C be E_C and V_C , respectively. Then, the algorithm proceeds as follows:

1. **Initialisation:** Initially create a subgraph with the only node as v . The neighbouring vertices of v are placed in the set U .

2. **Addition step:** Now iterate over the vertices in U and add those vertices to C that increase the modularity value of the subgraph C . This can be achieved by adding those neighbours that have a higher degrees.
3. **Deletion step:** Remove those vertices in each iteration from V_C such that the network remains connected and the subgraph modularity value increases.
4. **Final step:** In this step, we add neighbours of those vertices, which are left in the subgraph C , considering only those neighbours that are not already in U .

At the end of the algorithm, the subgraph with maximum subgraph modularity will be found. But there can be a chance where the source vertex can be removed from the network in the deletion step. In this case, no community will be found for source vertex v .

The time complexity of the aforementioned algorithm is $O(K^2d)$, where K is the number of vertices to be discovered, and d is the average degree of these vertices.

4.7 COMMUNITY DETECTION VS COMMUNITY SEARCH

The community search (CS) problem aims to find a community that contains the query node in a network. This can be thought of as a query dependent version of the community detection problem. For the toy example shown in Figure 5.29, the community of vertex A (constituent nodes in that community) is returned. Consider the case of the Facebook network where the number of nodes is in the order of billions. Therefore, finding the community is time taking and not feasible as such networks are also continuously evolving. In such cases, we are only interested in the community structure of specific vertices at a time. Therefore, we can use CS algorithms. All the CD algorithms are also applicable to CS. The most common algorithms used for CS are based on the following methods:

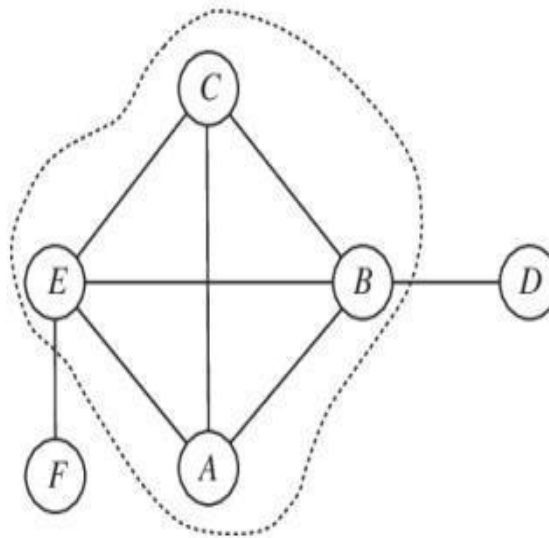


FIGURE 5.29. A toy example showing the CS node *A*, which will return other members of that community, i.e., *{B, C, E}*.

1. **K-core:** As discussed earlier, K-core is the maximal network in which each node has a minimum degree k .
2. **K-clique:** As discussed earlier, K-clique is a set of K vertices such that each pair of vertices has an edge between them.
3. **K-truss:** K-truss is the maximal subgraph such that every edge of the network is contained in at least $(K - 2)$ triangles in the subgraph.
4. **K-ECC (K-edge connected component):** K-ECC is a subgraph such that after removing any $K - 1$ edge, the subgraph is still connected.

CS problem is related to CD problem, but they are also different in the following ways:

- In CD, we find all the communities inside a network; in CS, we only see the community related to the query vertex. This is extremely useful in the case of social networks where billions of nodes are present. This is because, in such cases, CD is costly, time-consuming and often not necessary.
- In CD, we use global parameters to find communities in the network; in CS, the query parameter given by the user is used to find the communities in the network. This gives more personalised communities to the user, which in turn helps them to interpret the community structure easily.

- CD algorithms are generally time-consuming and non-scalable for large networks. Therefore, they cannot be used for online tasks. However, this is not the case for CS algorithms as they are meant to work for online study. The reason for being fast is that these algorithms are not applied to the entire network. Instead, they are used to the subgraph, which makes these algorithms more efficient.
- It is hard to use indexes and dynamic networks with CD; however, it is easy to do so with CS algorithms. As CD algorithms use global criteria, they are not sensitive to the node and edge updates. CS algorithms work well on continuously evolving networks.

4.8 EVALUATION OF COMMUNITY DETECTION METHODS

After applying the CD methods, our next task is to evaluate the quality of the obtained communities, i.e., CD methods. Evaluation becomes more manageable if we have a ground-truth community structure (the actual/true community structure of the network). If the resultant community structure returned by an algorithm highly resembles the ground-truth community structure, it can be considered a good CD algorithm. The following section discusses various methods that can be used to measure the quality of CD algorithms.

4.8.1 Ground-truth Based Evaluation

Here, we discuss various methods that can be used to measure the performance of CD algorithms in cases when the ground-truth communities are available.

Evaluation for Disjoint Communities

In this section, we discuss various metrics used to evaluate disjoint communities.

Purity. For a given network $G(V, E)$, let $C = \{c_1, c_2, c_3, \dots, c_K\}$ be the set of ground-truth community labels and $\Omega = \{\omega_1, \omega_2, \omega_3, \dots, \omega_J\}$ be the set of predicted community labels. Let $N = |V|$ be the total number of nodes in G . Given this information, the purity score can be obtained as,

$$\text{Purity}(C, \Omega) = \frac{1}{N} \sum_k \max_j |\omega_j \cap c_k| \quad (5.29)$$

Purity ranges from 0 to 1, and its value is maximum when the detected community structure is exactly the same as the ground-truth community structure. The problem with

purity is that we also get a value of 1 when obtained N singleton communities (each node is in each community).

Example 5.12

Purity of the community distribution shown in Figure 5.30.

$$\text{Purity} = \frac{\max(4,1,0) + \max(0,2,3) + \max(0,5,1)}{16} = 0.75 \quad (5.30)$$

$$Purity = \frac{\max(4, 1, 0) + \max(0, 2, 3) + \max(0, 5, 1)}{16} = 0.75$$

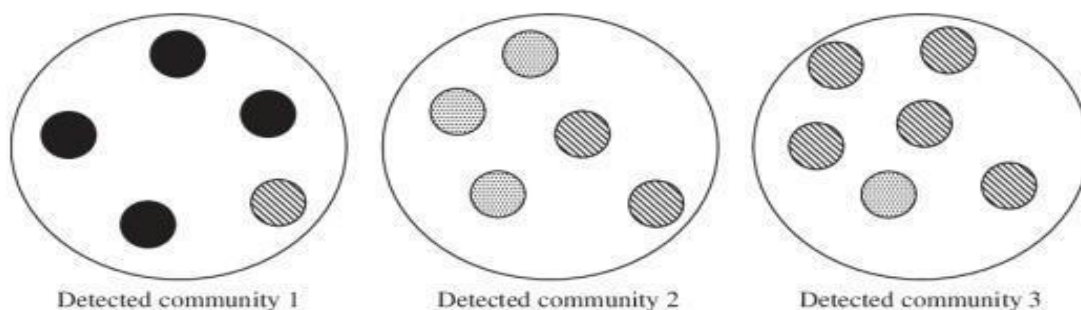


FIGURE 5.30. Comparing the ground-truth and detected community structures. Ground-truth communities are marked with three types of shades, and detected communities are marked with three circles.

Rand index. There are some basic metrics that are essential for evaluating the performance of CD methods (or that matter, any prediction framework). In the usual scenarios, these metrics indicate the nature of a “decision” taken by an algorithm. In the case of CD, this “decision” is with regards to a pair of nodes. Hence, we define the following terms:

1. **True positive (TP):** A pair of nodes are in the same community in the predicted and the ground-truth community structures.
2. **True negative (TN):** A pair of nodes are not in the same community in the predicted and ground-truth community structures.
3. **False positive (FP):** A pair of nodes are in different communities in the ground-truth community structure but in the same community in the predicted community structure.

4. **False negative (FN):** A pair of nodes are in the same community in the ground-truth community structure but in different communities in the predicted community structure.

What our algorithm should try to do is to maximise the values of TP and TN, and minimise the values of FP and FN. With this motivation, Rand index (RI) is defined, which measures the percentage of correct assignments. Mathematically, RI is given by,

$$RI = \frac{TP + TN}{TP + TN + FP + FN} \quad (5.31)$$

Example 5.13

Calculate the RI of the example shown in Figure 5.30.

Here,

$$TP = \binom{4}{2} + \binom{2}{2} + \binom{3}{2} + \binom{5}{2} = 20 \quad (5.32)$$

$$FN = \binom{4}{1} \binom{1}{1} + \binom{2}{1} \binom{3}{1} + \binom{5}{1} \binom{1}{1} + \binom{5}{1} \binom{1}{1} = 20 \quad (5.33)$$

$$TP + FP = \binom{5}{2} + \binom{5}{2} + \binom{6}{2} = 35 \quad (5.34)$$

$$TP + FP + FN + TN = \binom{16}{2} = 120 \quad (5.35)$$

$$FN + TN = 120 - 35 = 85 \quad (5.36)$$

$$TN = 85 - 20 = 65 \quad (5.37)$$

Therefore, RI is given as,

$$RI = \frac{65 + 20}{120} = 0.7083 \quad (5.38)$$

RI does not give us a correct evaluation of performance when the number of predicted communities and their sizes vary drastically from the ground-truth labels. The Adjusted Rand Index (ARI) was introduced by Santos and Embrechts, (2009). It is more popular in the field of CD. This metric is less susceptible to the number of communities. Mathematically, ARI is defined as,

$$ARI(\Omega, C) = \frac{\sum_j \binom{N_{\omega_j c_k}}{2} - [\sum_j \binom{N_{\omega_j}}{2} \sum_k \binom{N_{c_k}}{2}] / \binom{N}{2}}{[\sum_j \binom{N_{\omega_j}}{2} + \sum_k \binom{N_{c_k}}{2}] - [\sum_j \binom{N_{\omega_j}}{2} \sum_k \binom{N_{c_k}}{2}] / \binom{N}{2}}$$

$N_{\omega_j c_k}$ is the number of nodes in the set $\omega_j \cap c_k$.

Mutual information. It is given by

$$I(\Omega, C) = \sum_j \sum_k \frac{|\omega_j \cap c_k|}{N} \log \frac{N |\omega_j \cap c_k|}{|\omega_j| \cdot |c_k|}$$

The value generated by mutual information lies between 0 and 1. This value is maximum when both the detected and ground-truth communities have a similar structure. You might notice that I also go to 1 when each node is in its community. To solve this problem, normalised mutual information (NMI) is proposed, which is given as,

$$NMI(\Omega, C) = \frac{I(\Omega, C)}{\frac{1}{2} [H(\Omega) + H(C)]}$$

where H is the entropy,

$$H(\Omega) = - \sum_j \frac{|\omega_j|}{N} \log \frac{|\omega_j|}{N}$$

The problem with NMI is that it does not satisfy some popular properties of a metric such as the triangle inequality.

Evaluation for Overlapping Communities

Omega index. For a given network $G(V, E)$, let $C = \{c_1, c_2, c_3, \dots, c_K\}$ be the ground-truth community labels and $\Omega = \{\omega_1, \omega_2, \omega_3, \dots, \omega_J\}$ be the set of predicted community labels. Let N be the total number of nodes in the network. For this, we consider $|\omega_{ij}|$ and $|c_{ij}|$ as the number of communities shared by nodes i and j in the predicted communities and ground-truth communities, respectively. Then omega index is given as,

$$\Omega(\Omega, C) = \frac{1}{N^2} \left(\sum_{u,v \in V} 1 \{ |\omega_{uv}| = |c_{uv}| \} \right)$$

Overlapping normalised mutual information (ONMI). This is an extended version of mutual information for the overlapping communities. Let Y be the ground-truth community assignment, and X be the detected community assignment. The conditional probability of X given Y can be written as (this specifies lack of information for Y given that we know X),

$$H(X_k|Y_l) = H(X_k, Y_l) - H(Y_l) \quad (5.39)$$

$$H(X_k|Y) = \min_{l \in \{1,2,3,4,\dots,|C|\}} H(X_k, Y_l) \quad (5.40)$$

As this value is not normalised, we normalise this value and aggregate over all the values for X .

$$H(X|Y) = \frac{1}{|C|} \sum_k \frac{H(X_k|Y)}{H(X_k)} \quad (5.41)$$

The final ONMI value for the network can be given as,

$$N(X|Y) = 1 - \frac{1}{2} [H(Y|X) + H(X|Y)] \quad (5.42)$$

For a disjoint community, it can be seen easily that ONMI reduces to NMI.

F1-score. It is the average of the F1-scores of the best-matching detected community to each ground-truth community, and the average of the F1-score of the ground-truth community to the corresponding best-matching detected community (Yang and Leskovec 2013),

$$F1 = \frac{1}{2} \left(\frac{1}{|\Omega|} \sum_{a \in \Omega} F1(a, c_x) + \frac{1}{|C|} \sum_{b \in C} F1(\omega_y, b) \right)$$

where best matching x and y are defined as,

$$x = \arg \max_i F1(a, c_i)$$

$$y = \arg \max_j F1(\omega_j, b)$$

Here, a higher value of the F1-score implies a better algorithm that detects communities more accurately. F1-score can also be used to find the equivalence of two overlapping communities. For this, we need to find which $\omega_j \in \Omega$ corresponds to which $c_i \in C$.

Recently, Chakraborty et al. (2018) and Peel et al. (2017) argued against ground-truth based evaluation. Chakraborty et al. (2018) empirically showed that ground-truth based evaluation does not make sense as long as the meta-data of nodes is used for labelling the true community structure. With the change in the meta-data information, the entire true community structure will also change; however, the underlying topological structure will remain the same. The authors showed the evaluation of CD methods based on some specific tasks such as link prediction, message passing. They concluded that certain types of CD methods perform better for certain tasks.

