Social and Graph Data Management: Introduction to Data Models and Measures

Pablo Mollá Chárlez

Contents

1 Introduction

A graph community in a network is a group of nodes that are more densely connected to each other than to nodes outside the group. These communities, also known as clusters or modules, represent substructures within the larger network where members interact or collaborate more frequently among themselves. Identifying communities helps in understanding the organization, functionality, and dynamics of complex networks, such as social groups within a social network, functional modules in biological networks, or related topics in information networks. Some main features of communities include:

- **High Internal Connectivity**: Nodes within the same community have numerous connections among themselves.
- **Low External Connectivity**: There are relatively fewer connections between nodes of different communities.
- **Functional or Structural Similarity**: Members of a community often share common attributes, behaviors, or roles within the network.

Some examples of communities include Social Networks (groups of friends or colleagues who interact more frequently with each other than with others in the network), Biological Networks (functional modules such as protein complexes or metabolic pathways where components work closely together) or Information Networks (clusters of related web pages or articles that cover similar topics).

2 Fundamental Hypotheses

2.1 Hypothesis 1: Topological Encoding of Community Structure

Hypothesis 1 posits that a graph's community structure is uniquely encoded in its topology. This means that the arrangement and connectivity of nodes inherently contain the information necessary to discern distinct communities without additional metadata.

2.2 Hypothesis 2: Locally Dense Subgraph

Hypothesis 2 defines a community as a locally dense connected subgraph within a network. In other words, a community is a subset of nodes that exhibit a higher density of connections among themselves compared to their connections with the rest of the network.

2.3 Hypothesis 3: Absence of Community Structure in Random Networks

Hypothesis 3 asserts that random networks lack a community structure. Unlike structured networks where communities emerge due to specific interaction patterns, random networks distribute connections uniformly, resulting in no discernible clusters.

2.4 Hypothesis 4: Optimal Community Structure via Maximum Modularity

Hypothesis 4 suggests that the partition with maximum modularity corresponds to the optimal community structure. Modularity measures the strength of division of a network into communities, and the hypothesis claims that the highest modularity partition best represents the inherent community organization.

3 Community Detection Approaches

Detecting communities within a graph involves various strategies, each leveraging different properties of the network's topology.

- **Maximum Cliques**: One fundamental approach is identifying **Maximum Cliques**, where a community is defined as a subgraph in which every pair of nodes is directly connected. In this context, a clique represents the most tightly-knit group of nodes, with maximum internal connectivity.
- **Strong Communities** offer a relaxation of the clique concept by considering both internal and external degrees of nodes. Specifically, a strong community is characterized by nodes having a higher internal degree (number of neighbors within the community) compared to their external degree (neighbors outside the community). This balance ensures that the community is densely connected internally while maintaining sparse connections externally.

3.1 Naïve Algorithm for Community Detection

A straightforward method for detecting two communities involves:

1. **Finding a Cut**: Divide the graph into two subsets by identifying a cut that separates the nodes.
2. **Evaluating Strength**: Determine if the resulting partitions qualify as strong communities based on internal and external degrees.
3. **Selecting the Best Cut**: Among all possible cuts, choose the one that optimizes the community strength criteria.

However, this approach becomes computationally infeasible for larger graphs as it requires evaluating an exponential number of possible cuts, specifically 2^{N-1} cuts for a graph with N nodes. This exponential growth renders the naïve algorithm impractical for networks of substantial size.

3.2 Polynomial-Time Algorithms for Community Detection

To efficiently detect communities in larger networks, **polynomial-time algorithms** are essential. One prominent category is **Hierarchical Clustering**, which can be subdivided into agglomerative and divisive methods.

3.2.1 Hierarchical Clustering

Hierarchical clustering uses a similarity matrix X, where each entry x_{ij} quantifies the similarity between nodes i and j. Based on this matrix, communities are identified through either merging similar nodes (agglomerative) or splitting dissimilar ones (divisive).

3.2.1.1 Agglomerative Algorithms: The Ravasz Algorithm

An example of an agglomerative algorithm is the **Ravasz Algorithm**, which operates as follows:

1. **Define the Similarity Matrix**: Using the topological overlap matrix, where x_{ij} represents the number of common neighbors between nodes i and j relative to the maximum possible.
2. **Define Group Similarity**: Calculate the similarity between groups as the average similarity of all node pairs within the groups.
3. **Apply Hierarchical Clustering**:
   (a) Assign each node to its own community.
   (b) Identify and merge the pair of communities with the highest similarity.

(c) Update the similarity measures between communities.

(d) Repeat until all nodes are consolidated into a single community.

4. **Dendrogram Representation**: The sequence of mergers is visualized in a dendrogram, illustrating the hierarchical community structure.

3.2.1.2 Example of Ravasz Algorithm

Let us consider the similarity between nodes using topological overlap formula which is defined as follows:

$$X_{i,j} = \frac{|N(i) \cap N(j)|}{\min(|N(i)|, |N(j)|)}$$

This will allow us to build the similarity matrix for the graph. The graph is composed of the following nodes and edges:

- **Nodes**: A, B, C, D, E, F
- **Edges**: $A \longleftrightarrow B$, $A \longleftrightarrow C$, $B \longleftrightarrow C$, $C \longleftrightarrow D$, $D \longleftrightarrow E$, $D \longleftrightarrow F$, $E \longleftrightarrow F$.

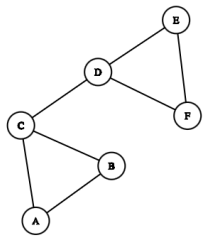

Figure 1: Network Example

Therefore, the neighborhood sets are as follows: $N(A) = \{B, C\}$, $N(B) = \{A, C\}$, $N(C) = \{A, B, D\}$, $N(D) = \{C, E, F\}$, $N(E) = \{D, F\}$,$N(F) = \{D, E\}$. For instance, computing the similarity between node A and node B, would be:

- $N(A) = \{B, C\}, \quad N(B) = \{A, C\}$.
- $N(A) \cap N(B) = \{C\}, \quad |N(A) \cap N(B)| = 1$.
- $|N(A)| = 2, \quad |N(B)| = 2$, so $\min(|N(A)|, |N(B)|) = 2$.
- $X_{A,B} = \frac{|N(A) \cap N(B)|}{\min(|N(A)|, |N(B)|)} = \frac{1}{2} = 0.5$.

By computing all pairwise similarities, we obtain the final similarity matrix.

$i \backslash j$	A	B	C	D	E	F
A	1	0.5	0.5	0	0	0
B	0.5	1	0.5	0	0	0
C	0.5	0.5	1	0.33	0	0
D	0	0	0.33	1	0.5	0.5
E	0	0	0	0.5	1	0.5
F	0	0	0	0.5	0.5	1

Now, we need to apply the Ravasz Algorithm to determine the communities within the graph.

- Step 1: Initialization We start with each node as its own community: $\{A\}, \{B\}, \{C\}, \{D\}, \{E\}, \{F\}$

- Step 2: Find the Highest Similarity Pair From the similarity matrix, we observe that the **highest similarity value** (excluding diagonal elements, which are always 1) is **0.5**. This value occurs for several pairs:
  - $(A, B), (A, C), (B, C)$
  - $(D, E), (D, F), (E, F)$

- Step 3: Process Each Pair in Order
  1. **First Pair** (A, B): We merge A and B into one community:
  $$\{A, B\}, \{C\}, \{D\}, \{E\}, \{F\}.$$
  2. **Next Pairs** (A, C) **and** (B, C): Since A and B are now in the same community, and the similarity between the community $\{A, B\}$ and node C is:
  $$X_{\{A,B\},C} = \frac{X_{A,B} + X_{A,C}}{\text{number of pairs compaired}} = \frac{0.5 + 0.5}{2} = 0.5.$$
  Therefore, we merge C in the community $\{A, B\}$ resulting in:
  $$\{A, B, C\}, \{D\}, \{E\}, \{F\}.$$
  3. **Next Pair** (D, E): We merge D and E into one community:
  $$\{A, B, C\}, \{D, E\}, \{F\}.$$
  4. **Next Pairs** (D, F) **and** (E, F): Following the previous procedure, we merge D, E, and F:
  $$\{A, B, C\}, \{D, E, F\}.$$

- Step 4: Update the Similarity Matrix After these merges, we compute the updated similarity matrix between the new communities. The similarity between communities is calculated as the average similarity between all node pairs in the two communities. From the original similarity matrix:
$$\begin{aligned} &S(A, D) = 0, \quad S(A, E) = 0, \quad S(A, F) = 0 \\ &S(B, D) = 0, \quad S(B, E) = 0, \quad S(B, F) = 0 \\ &S(C, D) = 0.33, \quad S(C, E) = 0, \quad S(C, F) = 0. \end{aligned}$$
And averaging the similarities:
$$\text{Average similarity} = \frac{\text{Total similarity}}{\text{Number of pairs}} = \frac{0 + 0 + 0 + 0 + 0 + 0 + 0.33 + 0 + 0}{9} = \frac{0.33}{9} \approx 0.037.$$
This computations, yield the following new similarity matrix:

$i \backslash j$	$\{A, B, C\}$	$\{D, E, F\}$
$\{A, B, C\}$	1	0.037
$\{D, E, F\}$	0.037	1

- Step 5: Resulting Communities At the end of the iterations, we have two clear communities:
$$\text{Communities: } \{A, B, C\}, \{D, E, F\}.$$

3.2.1.3 Divisive Algorithms: The Girvan-Newman Algorithm

The **Girvan-Newman Algorithm** exemplifies a divisive approach. The purpose of the algorithm is to detect communities in a network by iteratively removing edges that are considered the most "central" (based on **betweenness centrality**). The idea is that edges connecting different communities will have high centrality, and by removing these edges, the network will eventually split into separate communities. This algorithm helps reveal the modular structure of a network by identifying groups of nodes that are more tightly connected internally than with the rest of the network.

1. **Define Centrality**: Calculate betweenness centrality for all edges, identifying edges that bridge different communities.
2. **Apply Hierarchical Clustering**:
   (a) Remove the edge with the highest betweenness centrality.
   (b) Recompute betweenness centrality for the remaining edges.
   (c) Repeat the removal process until no edges remain.
3. **Dendrogram Representation**: The order of edge removals is captured in a dendrogram, revealing the emerging community structure as the graph fragments.

The Girvan-Newman algorithm generates an exponential number of possible cuts due to the iterative removal of edges, which significantly impacts computational complexity, especially for large networks.

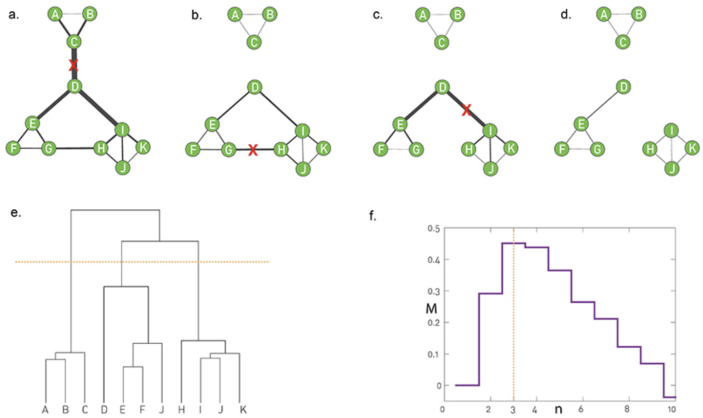

Figure 2: Girvan-Newman Algorithm

4 Modularity in Community Detection

4.1 Definition and Calculation

Modularity is a metric that quantifies the quality of a particular division of a network into communities. It compares the density of edges inside communities to the expected density in a random network with the same degree distribution. The modularity for a partitioning into communities $\mathcal{C}$ is defined as:

$$M = \sum_{C \in \mathcal{C}} \frac{1}{2L} \sum_{(i,j) \in C} (A_{ij} - p_{ij})$$

where:

- A_{ij} is the adjacency matrix of the original graph.

- $p_{ij} = \frac{k_i \cdot k_j}{2L}$ represents the probability of an edge between nodes i and j in a randomized version of the network, preserving the degree distribution.
- L is the total number of edges in the network.

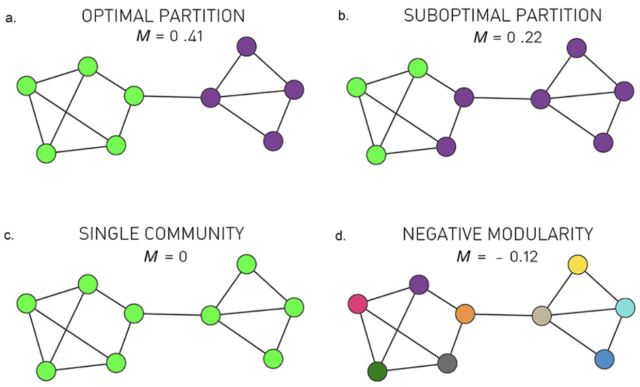

Figure 3: Modularity Example

4.2 Optimal Partition via Maximum Modularity

According to **Hypothesis 4**, the partitioning that maximizes the modularity score M corresponds to the optimal community structure. A higher modularity indicates a stronger community structure, where communities have more internal connections than expected by chance.

4.3 Modularity-Based Algorithms

Given the computational challenges of evaluating all possible partitions, efficient algorithms are necessary to approximate the optimal community structure.

4.3.1 Greedy Modularity Optimization

A common approach to maximizing modularity is the **Greedy Modularity Optimization** algorithm, which follows these steps:

1. **Initialization**: Assign each node to its own community.
2. **Iterative Merging**:
   (a) Inspect pairs of communities connected by at least one link.
   (b) Merge the pair that results in the highest increase in modularity ΔM.
3. **Termination**: Continue merging until all nodes belong to a single community.
4. **Selection**: Select the partition with the highest observed modularity during the merging process.

4.3.1.1 Example of Greedy Modularity

Let us consider a simple graph with 5 nodes and the following edges:

- **Nodes**: $\{A, B, C, D, E\}$
- **Edges**: $\{(A, B), (A, C), (B, C), (D, E)\}$

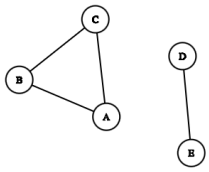

Figure 4: Modularity Graph

Then, we proceed as described in the steps of the algorithm:

1. Step 1: Initialize communities Initially, each node is its own community:

$$\text{Community } 1 : \{A\}, \quad \text{Community } 2 : \{B\}, \quad \text{Community } 3 : \{C\}$$
$$\text{Community } 4 : \{D\}, \quad \text{Community } 5 : \{E\}$$

2. Step 2: Compute modularity gain for all possible merges

   The goal is to find the pair of communities that, when merged, maximizes the modularity defined as:

$$\Delta M = \frac{1}{2m}\left(A_{AB} - \frac{k_A k_B}{2m}\right)$$

   Where:

   - A_{AB} is the number of edges between communities A and B,
   - k_A and k_B are the degrees of the nodes in the respective communities,
   - m is the total number of edges in the graph (which is 4).

   So, let's proceed by computing the modularity gain for all possible merges between pairs of communities.

   - Modularity gain for merging A and B:
     - Degree of $A = 2$, Degree of $B = 2$.
     - Number of edges between A and $B = 1$.
     - Expected number of edges between A and $B = \frac{2\times 2}{8} = 0.5$.
     - Modularity gain: $\Delta M = \frac{1}{8}(1 - 0.5) = 0.0625$.
   - Modularity gain for merging A and C:
     - Degree of $A = 2$, Degree of $C = 2$.
     - Number of edges between A and $C = 1$.
     - Expected number of edges between A and $C = \frac{2\times 2}{8} = 0.5$.
     - Modularity gain: $\Delta M = \frac{1}{8}(1 - 0.5) = 0.0625$.

- Modularity gain for merging A and D:
  - Degree of $A = 2$, Degree of $D = 1$.
  - Number of edges between A and $D = 0$.
  - Expected number of edges between A and $D = \frac{2\times1}{8} = 0.25$.
  - Modularity gain: $\Delta M = \frac{1}{8}(0 - 0.25) = -0.03125$.
- Modularity gain for merging A and E:
  - Degree of $A = 2$, Degree of $E = 1$.
  - Number of edges between A and $E = 0$.
  - Expected number of edges between A and $E = \frac{2\times1}{8} = 0.25$.
  - Modularity gain: $\Delta M = \frac{1}{8}(0 - 0.25) = -0.03125$.
- Modularity gain for merging B and C:
  - Degree of $B = 2$, Degree of $C = 2$.
  - Number of edges between B and $C = 1$.
  - Expected number of edges between B and $C = \frac{2\times2}{8} = 0.5$.
  - Modularity gain: $\Delta M = \frac{1}{8}(1 - 0.5) = 0.0625$.
- Modularity gain for merging B and D:
  - Degree of $B = 2$, Degree of $D = 1$.
  - Number of edges between B and $D = 0$.
  - Expected number of edges between B and $D = \frac{2\times1}{8} = 0.25$.
  - Modularity gain: $\Delta M = \frac{1}{8}(0 - 0.25) = -0.03125$.
- Modularity gain for merging B and E:
  - Degree of $B = 2$, Degree of $E = 1$.
  - Number of edges between B and $E = 0$.
  - Expected number of edges between B and $E = \frac{2\times1}{8} = 0.25$.
  - Modularity gain: $\Delta M = \frac{1}{8}(0 - 0.25) = -0.03125$.
- Modularity gain for merging C and D:
  - Degree of $C = 2$, Degree of $D = 1$.
  - Number of edges between C and $D = 0$.
  - Expected number of edges between C and $D = \frac{2\times1}{8} = 0.25$.
  - Modularity gain: $\Delta M = \frac{1}{8}(0 - 0.25) = -0.03125$.
- Modularity gain for merging C and E:
  - Degree of $C = 2$, Degree of $E = 1$.
  - Number of edges between C and $E = 0$.
  - Expected number of edges between C and $E = \frac{2\times1}{8} = 0.25$.
  - Modularity gain: $\Delta M = \frac{1}{8}(0 - 0.25) = -0.03125$.
- Modularity gain for merging D and E:
  - Degree of $D = 1$, Degree of $E = 1$.
  - Number of edges between D and $E = 1$.
  - Expected number of edges between D and $E = \frac{1\times1}{8} = 0.125$.
  - Modularity gain: $\Delta M = \frac{1}{8}(1 - 0.125) = 0.109375$.

3. Step 3: Select the merge with the highest modularity gain and merge the communities

   From the computed modularity gains, the highest gain is 0.109375 for merging D and E, therefore we merge them and obtain:

$$\text{Community } 1 : \{A\}, \quad \text{Community } 2 : \{B\}$$
$$\text{Community } 3 : \{C\} \quad \text{Community } 4 : \{D, E\}$$

4. Step 4: Repeat the process Now, we repeat the process of computing the modularity gain for merging the communities for all communities (single and pair communities). For instance:

   - Modularity gain for merging (D, E) and A:
     - Degree of $D = 1$, Degree of $E = 1$ ($\implies$ Degree of $\{D, E\} = 2$) , Degree of $A = 2$.
     - Number of edges between $\{A\}$ and $\{D, E\}$ is 0.
     - Expected number of edges between A and $D, E = \frac{2\times 2}{8} = 0.5$
     - Modularity gain: $\Delta M = \frac{1}{8}(0 - 0.5) = -0.0625$

   After computing all pairs of possible merges, we would select to merge $\{A\}$ with $\{B\}$ (or $\{B\}$ with $\{C\}$, $\{A\}$ with $\{C\}$ as they share the same modularity gain of 0.0625).

5. Step 5: Final merge decision The maximum modularity gain is 0.0625 for merging $\{A\}$ and $\{B\}$ or $\{A\}$ and $\{C\}$, so we choose one of these merges. Let's say we choose to merge $\{A\}$ and $\{B\}$, although in the next iteration where we compute the modularity gain we would add the node C to the community $\{A, B\}$, leading to the final communities:

$$\text{Community } 1 : \{A, B, C\}, \quad \text{Community } 2 : \{D, E\}$$

4.3.2 The Louvain Algorithm

The **Louvain Algorithm** offers an optimized method for modularity maximization with higher efficiency:

1. **Initialization**: Start with each node in its own community.
2. **Local Optimization**:
   (a) For each node, evaluate the gain in modularity by moving it to the community of each of its neighbors.
   (b) Assign the node to the community that provides the highest modularity gain.
3. **Aggregation**: Once no further individual node movements can increase modularity, aggregate nodes into their respective communities, creating a new network where each community is represented as a single node.
4. **Repeat**: Apply the local optimization and aggregation steps iteratively until no further modularity gains are possible.

The Louvain algorithm efficiently handles large networks by breaking down the problem into smaller, manageable subproblems.

5 Complexity of Community Detection Algorithms & Open Issues

5.1 Complexity of Community Detection Algorithms

The computational complexity of various community detection algorithms varies significantly:

Algorithm Type	Algorithm	Complexity
Agglomerative	Ravasz	$O(N^2)$
Divisive	Girvan-Newman	$O(N^3)$
Greedy Optimized	Modularity	$O(N \cdot \log^2 N)$
Modular Louvain	Louvain	$O(L)$
Flow-Based	Infomap	$O(N \cdot \log N)$

Table 1: Complexity of Community Detection Algorithms

Here, N denotes the number of nodes and L represents the number of edges in the network. The table highlights the scalability of algorithms, with Louvain and Infomap being more suitable for large-scale networks due to their lower computational complexity.

5.2 Open Issues in Community Detection

Despite advancements in community detection methodologies, several challenges remain:

- **Existence of Communities**: It is not always clear whether a given network inherently possesses a community structure.
- **Validity of Hypotheses**: The foundational hypotheses may not universally apply, questioning whether community structures are solely determined by topology.
- **Membership Ambiguity**: Determining whether every node belongs to a distinct community or can belong to multiple communities remains unresolved.
- **Selection of Measures**: With various metrics available (centrality, similarity, modularity, flow), identifying the most appropriate measure for a specific network context is challenging.

6 Conclusion

Community detection in graph theory is a multifaceted problem with significant implications across various fields. The hypotheses and methodologies discussed provide a framework for understanding and identifying community structures within networks. While algorithms like Girvan-Newman and Louvain offer effective means to detect communities, challenges such as computational complexity and the inherent nature of communities persist. Ongoing research continues to address these issues, striving for more accurate and efficient community detection techniques.