VIET NAM NATIONAL UNIVERSITY HO CHI MINH CITY
UNIVERSITY OF TECHNOLOGY
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

MATHS FOUNDATION FOR COMPUTER SCIENCE

ASSIGNMENT REPORT

COMMUNITY STRUCTURE IDENTIFICATION

Professor:	Nguyễn An Khương	
Members:		
	Lê Hoàng Anh Tài	2370761
	Nguyễn Duy Hùng	2370563
	Trịnh Hoàng Xuân Hãn	2370562
	Lưu Nguyễn Bảo Ngọc	2370565
	Nguyễn Văn Hoàng Khang	2110242
	Nguyễn Văn Khoa	2370499

Ho Chi Minh City, 03/2024

Contributions report

Member	Student ID	Contributions	Workload
Lê Hoàng Anh Tài	2370761	16.7%	100%
Nguyễn Duy Hùng	2370563	16.66%	100%
Trịnh Hoàng Xuân Hãn	2370562	16.66%	100%
Lưu Nguyễn Bảo Ngọc	2370565	16.66%	100%
Nguyễn Văn Hoàng Khang	2110242	16.66%	100%
Nguyễn Văn Khoa	2370499	16.66%	100%

Link Youtube: *Presentation video*

Table of Contents

1 Introduction

Community Structure Identification is a fundamental concept in network analysis that seeks to reveal the inherent organization of complex networks by pinpointing clusters of nodes that exhibit strong internal connectivity while being sparsely connected to nodes in other clusters. This approach is pivotal for dissecting the intricate web of relationships within networks across various domains, including social networks, biological networks, and technological networks.

Understanding the community structure of networks is vital for several reasons. Firstly, it provides insights into the natural organization of complex systems, shedding light on how entities within the network interact and form cohesive groups. Secondly, it aids in the identification of functional modules or subgroups within networks, which can be crucial for understanding system dynamics and functionality. Thirdly, community structure identification facilitates the detection of influential nodes or entities that play key roles in shaping network dynamics, enabling targeted interventions or optimizations.

The applications of Community Structure Identification are vast and diverse. In social networks, it helps in understanding social dynamics, identifying communities of individuals with similar interests or affiliations, and predicting behavior within these communities. In biological networks, it assists in identifying functional modules within biological systems, elucidating complex interactions between genes, proteins, and other biomolecules. In technological networks, it aids in optimizing network architecture, detecting anomalies or malicious activities, and improving the efficiency of communication or information dissemination.

1.1 Motivation

Understanding the modular structure of networks and identifying communities within them serves as a crucial endeavor across various disciplines. The motivation behind studying community structure identification lies in its potential to unveil hidden patterns, provide insights into network dynamics, and simplify complex systems for analysis. By discerning communities within networks, researchers can gain a deeper understanding of the relationships between nodes, predict behavior, and uncover underlying organizational principles.

Moreover, community detection in large networks holds practical implications. In domains like the World Wide Web, it enables the identification of thematic clusters, aiding in search engine optimization and content organization. In biochemical or neural networks, it simplifies functional analysis by identifying groups of nodes with shared functionalities or roles. Thus, the motivation behind community structure identification extends beyond academic curiosity, encompassing practical applications that could enhance decision-making processes and optimize system performance.

1.2 Use-case: Customer network

Problem statement: Suppose that we are running an online bookstore. Customers may be interested in more than one type of book. They can find their interests in the latest fantasy thriller by Stephen King while sharing some of their pleasures for mathematical textbooks. Finding which types of books each customer invests in can help us improve our service through investigations in the detected communities. In this case, each book is represented by a vertex in a graph and an edge between two books is created if these two are bought by the same customer.

In the realm of customer network analysis, our assignment aims to employ advanced techniques of Community Structure Identification, with a particular focus on utilizing the Louvain and Girvan-Newman algorithms. Our dataset is derived from a collection of political books, and the primary objective is to unravel the intricate community structure embedded within the customer network. By leveraging this approach, we intend to discern cohesive groups of customers who share common political interests or affiliations.

1.2.1 Dataset

At this assignment, we will choose dataset Political books. Main characteristics:

- Content:
    - The dataset represents interactions between political books.

  - Each book is treated as a node in the network.
  - Edges between books indicate frequent co-purchasing by the same buyers, suggesting common interests or topics.

- Files:
  - **meta.dimacs10-polbooks**: Provides metadata about the network.
  - **out.dimacs10-polbooks**: Contains the adjacency matrix of the network in whitespace-separated values format. Each line represents an edge between two books.
    - ID of the source node (from node).
    - ID of the target node (to node).
    - (Optional) Weight or multiplicity of the edge.
    - (Optional) Timestamp of edges in Unix time.

1.2.2 Methodology: The Louvain algorithms

The Louvain method is an algorithm to detect communities in large networks. It maximizes a modularity score for each community, where the modularity quantifies the quality of an assignment of nodes to communities. This means evaluating how much more densely connected the nodes within a community are, compared to how connected they would be in a random network. The Louvain algorithm is a hierarchical clustering algorithm, that recursively merges communities into a single node and executes the modularity clustering on the condensed graphs.

1.2.3 Expected outcomes

Through the application of the Louvain algorithm to the political books dataset, we anticipate uncovering distinct communities within the customer network, each representing groups of customers with shared political interests or reading preferences. These communities may reveal insights into the formation of customer clusters based on ideological viewpoints, party affiliations, or specific political topics. Furthermore, identifying influential nodes or books within each community could inform targeted marketing strategies, personalized recommendations, and community detection efforts in customer-centric applications. Ultimately, this analysis will contribute to a deeper understanding of customer behavior and preferences in the realm of political literature, facilitating more effective engagement and communication strategies.

2 Preliminaries

Several methods have been proposed to address the challenge of identifying community structures within social network graphs. In this section, we will delve into the definitions related to social network graphs and their properties, providing concrete examples for foundational algorithms like the Louvain Algorithm and others such as Girvan-Newman.

2.1 Modelize a social network

Graph is considered to be the most effective data structure for social network modelling. It can help people define and visualize entities and relationships between entities in networks.
We define a graph G = (V,E), where V is set of vertices, or nodes and E is set of edges. If there exists an edge between two vertices u and v, we say that u and v have a relationship and their edge is denoted as $\{u, v\}$. We can modelize a social network as a graph by assigning entities to vertices and relationships between them to edges.
A undirected graph is usually used to modelize a social network since it is suitable with the essence of social network that the relationships are bidirectional. Moreover, there is no vertex which has relationship with itself, meaning that $\{u, v\} = 0$ and there is an unique edge between two vertices.

2.2 How to perform social network graph

In this report, we use adjacency matrix A to perform a graph. It is an |V| x |V|. The entry in row i, column j of matrix, denoted as a_{ij} is 0 if there is an edge between two vertices i and j, and 0 otherwise. By this representation, we can easily check whether there exists an edge betwwen any two vertices i and j.

2.3 Community and group in a graph

Community in a network graph consists of vertices which are closely connected to each other than individuals in another community. Entity in the same community share some similar attributes.

2.3.1 Metrics for dectecting community

To determine whether a subgraph (or group of entities) is a community, we define some definitions: **Give a subgraph C of G, |C| = n_C, |G| = n.** Then we denote:

- **Internal degree k_v^{int} for v ∈ C**: is the number of edges between vertex v with all vertices in subgraph C.
- **External degree k_v^{ext} for v ∈ C**: is the number of edges between vertex v with all vertices outside subgraph C.
- Number of **internal edge** E_{C-C} of subgraph C: is the number of edges of subgraph C.
- Number of **external edge** E_C of subgraph C: is the number of edges which has a vertex is in subgraph C. By then, we can have $E_C - E_{C-C}$ is set of edges connecting communities.

Now we introduce these definitions:

- **Intro-cluster density of subgraph C, denoted as** $\delta_{int}(C)$ is the ratio between number of edges in subgraph C and number of maximum possible edges subgraph C could have, has the formula:

$$\delta_{int}(C) = \frac{|E_{C-C}|}{n_C(n_C - 1)} \tag{1}$$

- **Inter-cluster density of subgraph C, denoted as** $\delta_{ext}(C)$: is the ratio between number of edges connecting s vertex in C and one that is not in C, and the maximum possible number of edges connecting a vertex in C and one that is not in C, has the formula:

$$\delta_{ext}(C) = \frac{|E_C| - |E_{C-C}|}{n_C(n - n_C)} \tag{2}$$

- **Average density of a graph, denoted as** $\delta(G)$: is the ratio between the number of edges in graph G and maximum possible number of edges G could have, has the formula:

$$\delta(G) = \frac{|E|}{n(n-1)/2} \tag{3}$$

If C is a community in the graph, we expect $\delta_{int}(C) > \delta(G)$ and $\delta_{ext}(C) < \delta(G)$. In another word, we want $\delta_{int}(C)$ larger than $\delta(G)$ as possible and $\delta_{ext}(C)$ less than $\delta(G)$. This results in the target of maximizing $\delta_{int}(C) - \delta_{ext}(C)$.
Example 1: Given a graph G = (V,E) as figure 1 below with set of vertices V = $\{1, 2, 3, 4, 5, 6, 7, 8\}$ and set of edges E = $\{\{1, 2\}, \{2, 3\}, \{3, 4\}, \{2, 4\}, \{1, 4\}, \{4, 5\}, \{5, 6\}, \{5, 7\}, \{7, 8\}\}$. Examine three subgraphs C_1, C_2, C_3 as figure below:

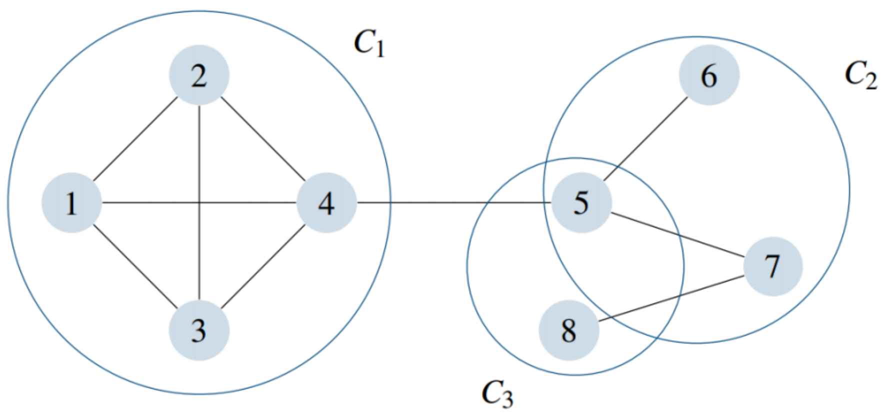

Figure 1: *Undirected Graph*

We estimate all metrics:

- Average density of graph G:

$$\delta(G) = \frac{|E|}{n(n-1)/2} = \frac{10}{8(8-1)/2} = \frac{5}{14} \approx 0.357$$

- Intra-cluster density of subgraph C_1:

$$\delta_{int}(C_1) = \frac{|E_{C_1-C_1}|}{n_{C_1}(n_{C_1}-1)/2} = \frac{6}{4(4-1)/2} = 1$$

- Intra-cluster density of subgraph C_2:

$$\delta_{int}(C_2) = \frac{|E_{C_2-C_2}|}{n_{C_2}(n_{C_2}-1)/2} = \frac{2}{3(3-1)/2} \approx 0.667$$

- Intra-cluster density of subgraph C_3:

$$\delta_{int}(C_3) = \frac{|E_{C_3-C_3}|}{n_{C_3}(n_{C_3}-1)/2} = \frac{0}{2(2-1)/2} = 0$$

- Inter-cluster density of subgraph C_1:

$$\delta_{ext}(C_1) = \frac{|E_{C_1}| - |E_{C_1-C_1}|}{n_{C_1}(n - n_{C_1})} = \frac{7-6}{4(8-4)} = \frac{1}{16} \approx 0.063$$

- Inter-cluster density of subgraph C_2:

$$\delta_{ext}(C_2) = \frac{|E_{C_2}| - |E_{C_2 - C_2}|}{n_{C_2}(n - n_{C_2})} = \frac{4-2}{3(8-3)} = \frac{2}{15} \approx 0.133$$

- Inter-cluster density of subgraph C_3:

$$\delta_{ext}(C_3) = \frac{|E_{C_3}| - |E_{C_3 - C_3}|}{n_{C_3}(n - n_{C_3})} = \frac{4-0}{2(8-2)} = \frac{1}{3} \approx 0.333$$

Based on the values that we just estimate, we have some reviews:

- $\delta_{int}(C_1) = 1 > \delta(G)$ and $\delta_{ext}(C_1) = 0.063 < \delta(G)$, so C_1 is likely a community.
- $\delta_{int}(C_2) = 0.667 > \delta(G)$ and $\delta_{ext}(C_2) = 0.133 < \delta(G)$, so C_2 is likely a community.
- $\delta_{int}(C_3) = 0 < \delta(G)$ and $\delta_{ext}(C_3) = 0.333 < \delta(G)$, so C_3 is not likely a community.

2.3.2 Overlapping between communities

An entity(or a node) can be a member of more than one community. This leads to variety in community structure. We can divide the community structure into three types:

- **Non-overlapping community**: refers to a community that is separate from others, where each member is exclusive to that community and does not participate in any other community.
- **An overlapping community**: is a type of community where a member can also be part of another community.
- **A nested community:** is a type of community that exists as a subset within a larger community.

2.4 Local graph and identifying community on local graph

A Local Graph is a smaller part of a larger graph, focusing on the relationships between one or a few vertices. In the context of social networks, a local graph is a subgraph where the set of vertices includes a central node, and other nodes, referred to as neighbor nodes, are connected to it. This local graph is utilized to depict the relationships of the central node.
The local graph has several characteristics:

- The central node is connected to other vertices by edges.
- Many vertices have a degree of 1, indicating that these vertices are connected only to the central node and no other vertices.
- All edges in the local graph hold significance to the central node.

In a local graph, communities exhibit little overlap, and the number of vertices within a community can vary.

2.5 Modularity Measure

Modularity measure is suggested by Newman and Girvan, computing the distribution of edges inside a community compared to a random graph with same edges' distribution. It is estimated by the formula:

$$Q = \frac{1}{2m} \sum_{ij} (A_{ij} - P_{ij}) \delta(C_i, C_j) \tag{4}$$

Where:

- A_{ij} is entry of row *ith* and column *jth* of adjacency matrix A of the graph.
- m is the number of edges

- P_{ij} is the expectation of number of edge between i and j
- δ function has the value 1 if i,j are in the same community, and 0 otherwise.

In notice, edge between two vertices i and j is the connection between two stubs. Given k_i, k_j are degree of i,j respectively, the probability of choosing one stub belong to i and j are $\frac{k_i}{2m}$ and $\frac{k_j}{2m}$ respectively. Since all edges are independent, the probability of connection between i and j is $p_i p_j = \frac{k_i k_j}{4m^2}$, which results in the expectation $P_{ij} = 2mp_i p_j = \frac{k_i k_j}{2m}$. So we have final formula of modularity is :

$$Q = \frac{1}{2m} \sum_{ij} (A_{ij} - \frac{k_i k_j}{2m}) \delta(C_i, C_j) \tag{5}$$

Note that δ is 0 when i and j are in different community. So the formula can be written:

$$Q = \frac{1}{2m} \sum_{C} \sum_{i,j \in C} (A_{ij} - \frac{k_i k_j}{2m}) \tag{6}$$

Since the total sum of all entries in adjacency matrix is sum of all degrees of vertices, also equal to $2l_c$ is number of edges in community C (Handshake Theorem), so:

$$Q = \frac{1}{2m} \sum_{C} \sum_{i,j \in C} \left(A_{ij} - \frac{k_i k_j}{2m} \right) = \sum_{C} \frac{l_C}{m} - \frac{1}{2m} \sum_{i,j \in C} \frac{k_i k_j}{2m} \tag{7}$$

If $i \equiv j$, then $k_i = k_j$ so

$$\frac{k_i k_j}{2m} = -\frac{k_i^2}{2m}$$

If $i \neq j$, then note that $\frac{k_i k_j}{2m} = \frac{k_j k_i}{2m}$ so we will have

$$Q = \sum_{C} \frac{l_c}{m} - \frac{1}{2m} \sum_{i,j \in C} \frac{k_i k_j}{2m} = \sum_{C} (\frac{l_c}{m} - \frac{(\sum_{i,j \in C} k_i)^2}{4m^2}) = \sum_{C} \left[\frac{l_c}{2m} - \left(\frac{d_c}{2m} \right)^2 \right] \tag{8}$$

Where

- C is a community
- l_c is number of edges in community C
- d_c is total degree of vertices in C

Example 2: Given the adjacency matrix of a graph in Example 1 below:

$$A = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Using the formula $Q = \sum_C \left[\frac{l_C}{2m} - \left(\frac{d_C}{2m} \right)^2 \right]$, we will have the number of edges l_1 of subgraph C_1 and total degree d_1 of subgraph C_1

$$l_1 = \frac{(\sum_{i \in C_1} (\sum_{j \in C_1}))}{2}$$

$$l_1 = \frac{(a_{11} + a_{12} + a_{13} + a_{14}) + (a_{21} + a_{22} + a_{23} + a_{24}) + ... + (a_{41} + a_{42} + a_{43} + a_{44})}{2}$$

$$l_1 = \frac{(0+1+1+1) + (1+0+1+1) + (1+1+0+1) + (1+1+1+0)}{2} = 6$$

$$d_1 = \sum_{i \in C_1} (\sum_{j \in E} a_{ij})$$
$$d_1 = (a_{11} + a_{12} + ... + a_{18}) + (a_{21} + ... + a_{28}) + ... + (a_{41} + ... + a_{48})$$
$$d_1 = (0+1+1+1+0+0+0+0) + ... + (1+1+1+0+1+0+0+0) = 13$$

Similarly, we can estimate $l_2 = 2, d_2 = 6, l_3 = 0, d_3 = 4$. Replace these value into the formula we will have

$$Q = \sum_C \left[\frac{l_C}{2m} - \left(\frac{d_C}{2m} \right)^2 \right]$$
$$Q = \frac{l_1}{2m} - \left(\frac{d_1}{2m} \right)^2 + \frac{l_2}{2m} - \left(\frac{d_2}{2m} \right)^2 + \frac{l_3}{2m} - \left(\frac{d_3}{2m} \right)^2$$
$$Q = \frac{6}{10} - \left(\frac{13}{20} \right)^2 + \frac{2}{10} - \left(\frac{6}{20} \right)^2 + \frac{0}{10} - \left(\frac{4}{20} \right)^2$$
$$Q = 0.31$$

3 Approaches

Before going further, note that the contribution of vertex i , assigned to a certain modularity α, to the modularity of a partition is given by

$$Q_i = \frac{1}{2m}\left(\sum_{j\in\alpha} A_{ij} - \frac{k_i k_\alpha}{2m}\right) \tag{9}$$

where k_α is the degree of community α, defined as the sum of the degrees of its vertices. This expression shows that the contribution of vertex i requires only local information, its number of neighbors inside the community, its degree and the degrees inside the community. Locality is essential to design efficient greedy algorithm for modularity optimization.

3.1 The Louvain method

The Louvain method is a greedy optimization heuristic originally proposed to optimize modularity, but its flexibility allows to optimize other quality functions as well, as described below. The approach consists in two phases that are iteratively repeated, until a maximum of modularity is obtained. The first phase consists in moving vertices from their community to that of their neighbors until a local maximum is reached. The second phase then consists in forming a new graph whose vertices are obtained by aggregating the vertices attached to the same community in the previous phase. A combination of the two phases is called a ”pass”. Starting from a network composed of n vertices, the number of vertices decreases at each pass, until no improvement of modularity can be obtained.

First phase: vertex mover (VM). The first phase begins with an undirected weighted graph having C vertices to which an index between 0 and $C - 1$ has been randomly assigned. C is the number of vertices in the first pass, and the number of communities found in previous pass otherwise. One starts by placing each vertex into its own community, that is we start with a partition made of C communities. We then consider the first vertex, i.e. with index 0, and evaluate the change of modularity by removing 0 from its community and placing it in the community of each of its neighbors. Importantly for the speed of the method, this step only requires local information – See Eq.(9) –, and its number of operations is proportional to the degree of the vertex. Vertex 0 is then assigned to the community where the increase is maximum, if this maximum increase is positive. Otherwise, vertex 0 is left in its original community – See Algorithm 1 (all algorithms are adapted from [1]). This step is repeated sequentially to all the vertices based on their index. When reaching vertex $C - 1$ the process restarts at vertex 0, and the iteration continues until no vertex is moved during a complete iteration over the C vertices – See Algorithm 2. The phase ends in a local maximum, where greedy move of the vertices between community do not allow for an increase of the modularity. At this moment, a new partition of the vertices into C' communities has been produced. If $C' \neq C$, one proceeds to the next phase, and update the value of the number of communities. Else, the algorithm is finished and returns the partition as its estimate for optimum of modularity.

Second phase: aggregation. The second phase consists in building a new weighted graph whose vertices are the C communities found during the first phase. The weight of the link between two communities is given by the sum of the weights of the links between the vertices of these two communities. Moreover, links between the vertices of a same community lead to self-loops in the aggregated network. This operation is performed to ensure that the modularity of a partition in the aggregated graph has the same value as that of the same partition on the original graph. Once the second phase is finished, the first phase is reapplied on the aggregated network.

The output of the algorithm is the partition uncovered in the last pass, as it is the best partition found in terms of modularity – See Figure 1 and Algorithm 3. The method can also provide intermediary steps, that is the set of partitions of each pass, which provides a hierarchical representation of the network, each community at a pass being composed of a combination of combinations at the previous pass. Since its inception, the method has consistently been shown to provide a good balance between accuracy and speed of execution, and to have a complexity close to linear in benchmarks and empirical networks. The speed of the method is ensured by the locality of each operation, as vertices only move to neighboring communities. Its accuracy arises from its flexibility, as vertices may be removed from their community and re-assigned to others in the process, and its multi-scale nature, as the community exploration is first done locally, and then over longer distance as the vertices are aggregated. Combined with its simplicity, these strengths have made the Louvain method a popular choice to optimize not only modularity, but also other quality functions, including the Map Equation [2] and Markov stability [3], as we discuss more in detail below. When applied on modularity, as is usually the case, it is important to remember that the Louvain method inherits the limitations of modularity, including its resolution limit.

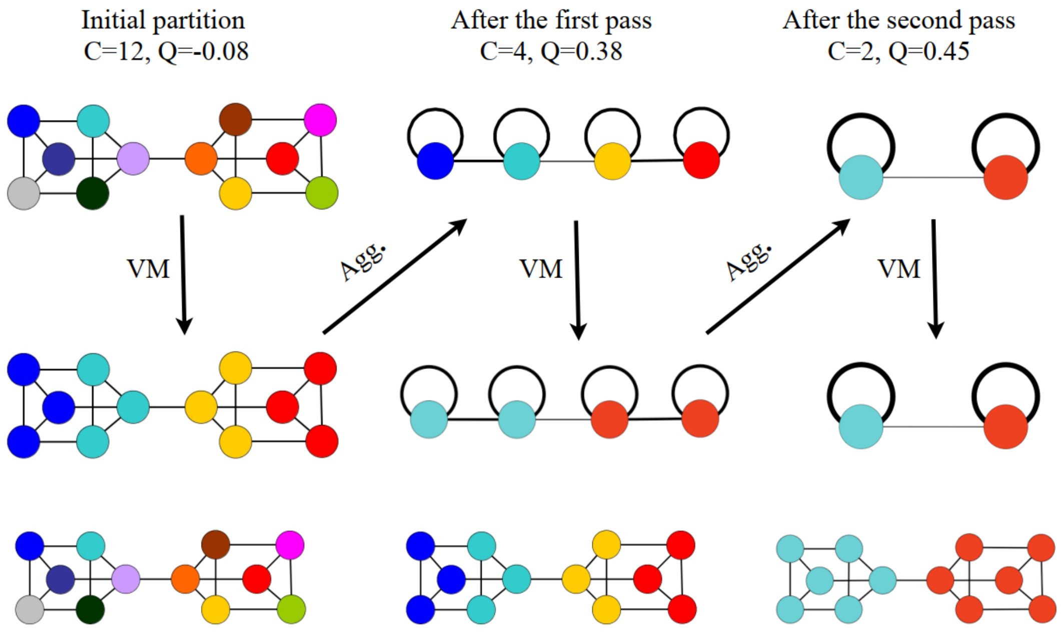

Figure 1: *Each pass of the Louvain method is made of two phases. The first phase (Vertex Mover) consists in sequentially moving vertices to neighboring communities leading to a maximum increase of modularity. The second phase aggregates vertices and constructs a meta-graph whose vertices are the communities found after the first phase. The two phases are repeated until no improvement of modularity is observed. In this illustration, the algorithm converges after two passes, uncovering an optimal partition made of 2 communities, and a value of* $Q = 0.45$. *In this example, it is easy to show that aggregating further the two communities into one big community would result in a decline of modularity, as* $Q = 0$ *in that case. Figure adapted from [4].*

Algorithm 1: SingleVertexMover

```
Require: $G = (V, E, w)$ a weighted graph
Require: $P$ a partition of $V$ ($P[i]$ = community of vertex $i$)
Ensure : $\Delta Q$, the gain of modularity
begin
    $c_{\text{old}} \leftarrow P[i]$
    REMOVE$(i, c_{\text{old}})$
    $C \leftarrow \{P[j] | (i,j) \in E\} \cup \{c_{\text{old}}\}$
    $c_{\text{new}} \leftarrow \arg\max_{c \in C}(\text{GAIN}(i, c))$
    if GAIN$(i, c_{\text{new}})$ − LOSS$(i, c_{\text{old}}) > 0$ then
        INSERT$(i, c_{\text{new}})$
        return GAIN$(i, c_{\text{new}})$ − LOSS$(i, c_{\text{old}})$
    else
        INSERT$(i, c_{\text{old}})$
        return 0
```

3.1.1 Louvain++, a selection of enhancements

Because of its effectiveness and simplicity, the Louvain method has been extensively studied, and numerous modifications have been proposed, whether to improve results, further increase calculation speed or generalize its scope of use. We present these modifications in four different directions, each detailed in its respective subsection:

- the modification of the algorithm's core, including its initialization, stopping criterion, vertex mover, etc;
- the use of quality functions other than modularity, functions which may be similar to modularity but with other parameters, or functions which may be completely different;

Algorithm 2: VertexMoverIteration

```
Require: $G = (V, E, \omega)$ a weighted graph
Require: $\epsilon$ a stopping criterion
Ensure : a partition $P$ of $V$
begin
    INIT($P$)
    do
        $increase \leftarrow 0$
        forall vertices $i$ do
            $increase \leftarrow increase$ + SingleVertexMover($i$)
    while $increase > \epsilon$
    return $P$
```

Algorithm 3: Louvain algorithm

```
Require: $G = (V, E, \omega)$ a weighted graph
Ensure : a partition $P$ of $V$
begin
    repeat
        $P \leftarrow$ VertexMoverIteration($G$)
        $G \leftarrow$ PartitionToGraph($P, G$)
    until no improvement is possible
```

- generalizations of Louvain to deal with more than simple graphs. These may include weighted, oriented, spatially constrained, temporal network, etc;
- and finally the use of parallelism to speed up computation time.

3.1.2 Modification of the algorithm

The original article and an extended version of it [4] (published in French in 2011 and in English in 2013) already described several improvements that have been proposed independently and studied in greater details by other authors:

- a partial optimization that consists in stopping the optimization when the modularity gain is below a given threshold (both for iterations or aggregation). This very simple optimization was already implemented in the initial version of Louvain;
- removing leaf vertices, which consists of deleting the leaves since they can only be placed in the community of their single neighbor;
- a VM that consists in not considering all vertices at each iteration;
- returning to lower levels to move again vertices;
- a simulated annealing-inspired VM allowing for non-optimal moves.

In the following, we describe modifications proposed in the different steps of the algorithm, that is its initialization, the vertex mover step and the aggregation step.

Initialization: Several types of optimization have targeted the initialization phase. The first one is to reduce the size of the graph before computation by removing a number of vertices that have little or no impact on the results. Based on the observation that a leaf (any vertex with a single neighbor) will inevitably be placed in its neighbor's community, several authors propose deleting these leaves [4, 5] or, more generally, all tree structures, i.e. the initial leaves and all those that will appear when deleting leaves [6]. The resulting graph corresponds to the 2-core of the original graph and the leaves are added back once the communities are found. A generalization has been proposed in [7] where a k-core decomposition is performed and communities are only computed on the k-core. All the removed vertices are then added back and assigned to a community. The authors propose to use label propagation or any modularity maximization algorithm for this task.

Other approaches have modified the initial partition by not starting with communities containing a single vertex. The authors of [8] identify cliques containing at least 3 vertices that serve as initial communities. However, the way cliques are identified and the management of overlaps between these cliques are not made explicit.

Vertex mover: In the Louvain method, the best choice is always preferred during the vertex mover operation. Based on the principle of simulated annealing, it was proposed to choose a non optimal move, even with a negative gain, with a low probability rather than the optimal one [9].

Several articles propose to avoid considering all vertices at each iteration. The simplest version is defined in [10]. If a vertex does not move for several iterations (with a fixed parameter) then it will not be considered anymore. In [6], the authors show that if two communities A and B have not changed in the last iteration, then no vertex can move from A to B or from B to A in the current iteration. A similar idea is used in [11] where the author identify vertices that are more likely to move in the next iteration and only consider these vertices. They propose four situations, the simplest of which considers that if a vertex i has just moved from the community A to the community B, then all i's neighbors who are not in B are likely to join i at the next iteration. The other three situations are a little more complex and are not used in practice. Limiting oneself to considering only a few vertices at the next iteration could lead to iterations being stopped prematurely and thus to a loss of quality, but in practice the impact is very slight.

In a different fashion, authors of [5] use a concept of seed vertices which are vertices more central than the average (for a given centrality measure defined in the paper). Non-seed vertices are then moved in priority to the neighboring community that contains a seed-vertex and maximizes the (positive) gain of modularity. If there is no such neighboring community or if no gain is positive, then the classical vertex mover is used.

In [12], instead of considering all neighbor communities and pick the best, one neighbor is picked at random and only its community is considered – other random selection procedure have also been considered, e.g. a community is selected proportionally to its size. Even though random selection will not always make the best choice, if a vertex has many neighbors in a given community then this community is more likely to be picked. The author notes that this operation accelerates convergence. This principle has been reused in the Leiden method [13] with other elements.

In most implementations of Louvain, the order in which vertices are considered is random. Although this can be seen as a problem, it does allow us to explore more possible solutions. If the aim is to obtain an identical partition, any fixed order can be used. In [14] the authors treat the vertices by decreasing centrality. This choice appears to improve results slightly, but above all, it speeds up the algorithm.

Aggregation and refinement: A key element of the Louvain method is that the vertex mover operation is not greedy since any given vertex can be moved several times until convergence is reached. As this step is not irreversible, it is less likely to be penalized by poor initial choices. However, the aggregation phases are irreversible, and the original method does not allow to go back once the graph has been aggregated. For this reason, a number of papers have proposed to challenge this principle and allow to move lower-level vertices even after aggregation. This procedure is called partition refinement.

In the simpler versions [15], a single refinement of the partition is performed on the first level, i.e., the original graph where all vertices are subject to the vertex mover procedure until convergence. In [16, 17], a multi-level refinement is used where, starting from the top level of the partition, the graph is unaggregated once and vertices are allowed to move. This process is repeated moving down the hierarchy until they are back on the initial graph. A different refinement is proposed in [18, 19] where the subgraph induced by each community is partitionned using one level or the complete Louvain method. All these parts of each communities are used to build the aggregated network. A more complex scheme is used in the Leiden method [13] where the vertex mover during refinement is limited and a move is accepted if it increases the modularity even though it is not the best move. Finally, in [20], if small (less than 4 vertices) or weak (that score low using their quality function) communities are found, these communities are broken into individual vertices and a VM step is applied once.

The Louvain method and its evolutions do not impose a priori constraints on the number or size of communities. If the number of communities is known to be k, then [21] propose to stop the aggregation in Louvain at the highest level that contains more than k communities (i.e. the next aggregation would result in fewer than k communities). They then use a greedy method to merge the communities that maximize the modularity gain to obtain exactly k communities. Finally, VM iterations are done until convergence.

3.1.3 Modification of the quality function

Despite the shortcomings of modularity, several quality functions have been inspired by it and used as a replacement in the Louvain method. For those, the algorithm remains unchanged after adapting the expression for the modularity change during the VM step. These quality functions mostly consider a positive contribution for intra-community edges and a negative contribution for inter-community edges or in the absence of edges.

- Several variations of modularity are using a resolution parameter, and can be shown to be equivalent, up to a trivial map, to the so-called Potts modularity [22]

$$Q_\gamma = \frac{1}{2m}\sum_{C}\sum_{i,j\in C}(A_{ij} - \gamma\frac{k_i k_j}{2m}) \tag{10}$$

  where the resolution parameter γ allows to give more or less importance to the negative term, and hence to modulate the size of the communities. This expression has the same optimum as the so-called Markov stability, where the quality of a partition is estimated by its tendency to capture random walkers for long times inside communities and where time plays the role of the resolution parameter. Most implementations ò Louvain have been generalized for use (or both) of these equivalent variations.

- The balanced modularity [23, 24] uses a symmetric version that also considers the absence of links

$$\frac{1}{2m}\sum_{i,j\in V}\left[\left(a_{ij} - \frac{k_i k_j}{2m}\right)x_{ij} - \left(\bar{a}_{ij} - \frac{(n-k_i)(n-k_j)}{n^2-2m}\right)\overline{x}_{ij}\right]$$

- The authors of [39] use a function that considers both edges within communities and edges between communities. The linear version of their function is

$$\frac{1}{2m}\sum_{i,j\in V}\left[\left(a_{ij} - \frac{k_i k_j}{2m}\right)x_{ij} - \beta\left(a_{ij} - \frac{k_i k_j}{2m}\right)^{\alpha}\overline{x}_{ij}\right]$$

  where α and β are resolution parameters that can be tuned to adjust the size of communities.

- In [24], the authors propose Jaccard Cosine Share Measure (JCSM). For two vertices i and j, the main term combines Jaccard (size of the intersection of neighborhoods of i and j divided by the size of their union), Cosine (normalized dot product of rows i and j of the adjacency matrix) and classical modularity $(a_{ij} - k_i kj/2m)$.

- Other functions mostly based on the number of connections within or between communities have been proposed. For instance [20] propose a new function defined as the sum for each community of

$$F2(C_i) = \frac{[d_{in}(C_i)]^2}{[d_{in}(C_i) + d_{out}(C_i)]^2}$$

- Based on other principles than modularity, the map equation introduced in [2] uses the length of description of random walks in a network. The equation uses a two levels description to encode both the modules and the vertices. vertices from different modules can use the same code (the authors use the analogy of street names, which may be similar in different cities). The optimization of the map equation follows closely the steps of the Louvain method. Note that the map equation has also been generalized to uncover a hierarchy of communities [19].

More generally, works have explored the conditions that need to be satisfied by a quality function so that it can be optimized by the Louvain method. In that direction, the authors of [1] have used relational coding to describe several quality function using the same formalism. A quality function F is said to be linear if $F(X, A) = \sum_{i,j \in V} \phi(a_{ij})x_{ij} + K$, where $a_{ij} = 1$ if i and j are connected (0 otherwise), ϕ and K are respectively a function and a constant that only depends on the original data. For example, modularity can be expressed as $Q = \sum_{i,j \in V}(a_{ij} - k_i kj/2m)x_{ij}$. If a quality function is linear then the gain obtained using VM can be computed locally, henceforth without loss of efficiency. The authors also study separable quality function, i.e. that can be written as $F(X, A) = \sum_{i,j \in V} \phi(a_{ij})\psi(x_{ij}) + K$, where ψ only depends on X. If the function ψ only depends on the community of interest then the corresponding quality function can also be used efficiently in Louvain. The authors study five non-linear but separable function and clarify why only some can be optimized efficiently. A different formulation is given in [25]. The Louvain method can be used to optimize any quality function of the form $trace\ H^T \left[F - ab^T\right] H$, where H is the partition indicator matrix, F is a general matrix derived from the network, and a and b are two n dimensional vectors.

3.1.4 Louvain for non-simple graphs

Modularity has originally been defined for undirected, unweighted networks. Since then, several generalizations have been proposed to allow for more complex graph structures. As compared to the expression Eq.(9), the first term often remains unchanged, to count the total number, or total weight, of edges inside communities, but the second term is modified to take into account the additional constrains induced by the graph structure. For instance, in the case of directed networks, the null model is now chosen according to the directed configuration model, to properly estimate the expected number of edges between two vertices, given their respective in-degree and outdegree [26]. The Louvain method can then be directly generalized with an adapted modularity gain during VM [27]. Other examples include the optimization of modularity for spatial networks where, again, the null model of the quality function can be modified in order to account for spatial constraints [28]. Alternative approaches keep standard expressions for modularity but modify instead the Louvain method to impose a spatial contiguity of the clusters in the uncovered partition [29].

Community detection is also an active field of study for temporal networks [30, 31]. Compared to its static counterpart, communities are now dynamical objects that may encounter different types of change, such as growth or splitting, and algorithms have the additional task to characterize these events. Two different types of approach build on community optimization. First, communities can be uncovered in different snapshots. In that case, some constraints on the communities in adjacent snapshots may be incorporated in order to ensure their continuity in time, such as in modularity maximization under estrangement constraint [32]. After the communities have been found in each snapshot, additional methods determine, with a certain statistical accuracy, when and how a reorganization takes place [33]. A second family of methods aims at finding a decomposition of the network in an extended network in which snapshots are concatenated, e.g. by connecting the same vertex in consecutive snapshots to favor continuity of communities across time. Here, instead of searching communities in each time window, the community detection is performed in one single operation, which can be formulated as a modularity maximization, on a larger, multilayer network [34] and again optimized by the Louvain method [35].

Modularity has also been generalized to quantify the quality of the partition of signed networks. Here, the intuition is place together vertices such that positive edges are concentrated within the modules and negative edges between the modules. In its simplest setting, the signed modularity consists in dividing the signed graph into two graphs, one defined by the positive interactions and another one defined by the negative interactions, and to search to maximize the difference of the modularities of the positive and negative graph [36]. The Louvain method can be directly generalized for this quality function, and the only step that needs to be considered with care is VM. Consider a vertex i. One expects to find other vertices of the community of i in the neighborhood of i only for the positive edges but its negative neighbors should instead be placed in other communities. Ways to address this complication are either to consider all the communities in the network in the VM step - and not only the neighboring ones - , but this involves a massive slow down of the algorithm, or to consider second neighbors, that is neighbors of neighbors, in the case of negative edges [37].

Finally, some works have focused on attributed networks where nodes carry labels. The aim is to create well-connected communities that are also homogeneous with regard to labels, in order to meet both structural and homophilic criteria. Three families of approaches are proposed in [38]: early fusion where a new graph is built to encode both topology and attributes, simultaneous fusion where the clustering is performed simultaneously on both domains and late fusion where both clusterings are computed independently before being joined. Among those, SAC2 [39] is an early fusion method where each node retains only its k-nearest neighbors based on a weighted combination of topology and similarity between attributes. Louvain is then used on this construction. Simultaneous fusion generally involves a composite function of modularity for the structural part and a similarity function for the attributes. This is the case in [40] using purity for attributes. Relatedly, SAC1 [39] uses different similarities for attributes, and [41] uses inertia. In all cases Louvain is used to optimize these functions. A late fusion is proposed in [42] where structural communities are found with Louvain and attributed ones with k-means. The structural communities are only used if they are sufficiently good, otherwise the attribute ones are used. More generally any consensus-based solution can be used to merge several partitions.

3.2 The Girvan-Newman algorithm

The Girvan-Newman algorithm works based on the idea that edges with high "betweenness" centrality are likely to be "bridges" between communities, and removing them will help to reveal the community structure. Specifically, the algorithm has two central features that distinguish it from those that have preceded. First, a "divisive" technique is used which iteratively removes edges from the network, thereby breaking it up in communities. The edges to be removed are identified by using one of a set of edge betweenness measures. Second, the algorithm includes a recalculation step in which betweenness scores are re-evaluated after the removal of every edge. This step turns out to be of primary importance to the success of its. Without it, the algorithms fail miserably at even the simplest clustering tasks.

The algorithm's steps for community detection are summarized below:

1. The betweenness of all existing edges in the network is calculated first.
2. The edge(s) with the highest betweenness are removed.
3. The betweenness of all edges affected by the removal is recalculated.
4. Steps 2 and 3 are repeated until no edges can be removed without disconnecting the network.

3.2.1 The betweenness calculation

The betweenness measure is some measure that favors edges that lie between communities and disfavors those that lie inside communities. Based on [43], there are 3 different measures corresponding to various implementations. However, for most problems, it is recommended to use the algorithm with betweenness scores calculated using the shortest-path betweenness measure:

- **Shortest-path betweenness**: This measure was first introduced by Anthonisse in a never published technical report in 1971 [44]. Anthonisse called it “rush”, but the authors prefer the term *edge betweenness*, since the quantity is a natural generalization to edges of the well-known (vertex) betweenness measure of Freeman [45]. This measure can be thought of in terms of signals traveling through a network. If signals travel from source to destination along geodesic network paths, and all vertices send signals at the same constant rate to all others, then the betweenness is a measure of the rate at which signals pass along each edge.

- **Random-walk betweenness**: The algorithm calculates the expected net number of times that a random walk between a particular pair of vertices will pass down a particular edge and sum over all vertex pairs. The random-walk betweenness can be calculated using matrix methods.

- **Current-flow betweenness**: This betweenness measure is motivated by ideas from elementary circuit theory. The authors consider the circuit created by placing a unit resistance on each edge of the network and unit current source and sink at a particular pair of vertices. The resulting current flow in the network will travel from source to sink along a multitude of paths, those with least resistance carrying the greatest fraction of the current. It is defined to be the absolute value of the current along the edge summed over all source/sink pairs. It can be calculated using Kirchhoff's laws. The entire community structure algorithm, including the recalculation step, will take $O((n+m)mn^2)$ time to complete, or $O(n^4)$ on a sparse graph. Although, as claimed in [43], the algorithm is good at finding community structure, this poor performance makes it practical only for smaller graphs; a few hundreds of vertices is the most that they have been able to do. It is for this reason that the authors recommended using the shortest-path betweenness algorithm in most cases, which gives results about as good or better with considerably less effort.

For this specific research use-case and with the brief summary above of each measure, we will only focus on how to calculate the **shortest-path betweenness** measure, which will be described as follows:

1. From n source vertices, select one vertex to be a starting root node X and perform breadth-first search to find number of shortest path from the node X to each node, and assign the numbers as score to each node.

2. Starting from the leaf nodes, to the edge from vertex i to vertex j, with j being farther from the root node X than i, we calculate the weight of edge by 1 plus the sum of the weights on the neighboring edges immediately below it, all multiplied by $\frac{w_i}{w_j}$, with w_i and w_j are the assigned scores of i node and j node, respectively.

3. Compute the weights of all edges in the graph, and repeat from step 1 until all of the source vertices are selected.

4. Sum up all of the edge weights we compute in step 2 of n selected vertices and divide by 2, and the result is the edge betweenness of edges.

At first sight, it appears that calculating the edge betweenness measure based on geodesic paths for all edges will take $O(mn^2)$ operations on a graph with m edges and n vertices: calculating the shortest path between a particular pair of vertices using breadth-first search can be done in time $O(m)$ [46, 47], and there are $O(n^2)$ vertex pairs. Recently however new algorithms have been proposed by Newman [48] and independently by Brandes [49] that can perform the calculation faster than this, finding all betweennesses in $O(mn)$ time. Both Newman and Brandes gave algorithms for the standard Freeman vertex betweenness, but it is trivial to adapt their algorithms for edge betweenness. Breadth-first search can find shortest paths from a single vertex s to all others in time $O(m)$. In the simplest case, when there is only a single shortest path from the source vertex to any other, the resulting set of paths forms a shortest-path tree - see Fig.2a. In general, however, it is not the case that there is only a single shortest path between any pair of vertices. Most networks have at least some vertex pairs between which there are several geodesic paths of equal length. Figure 2b shows a simple example of a shortest path "tree" for a network with this property.

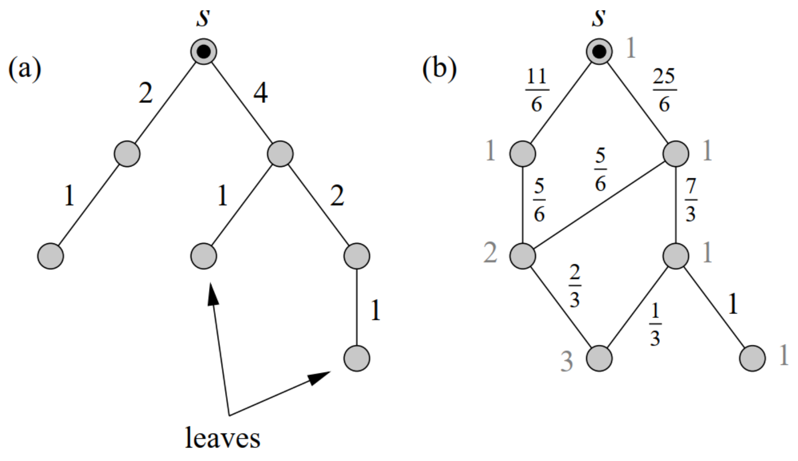

Figure 2: *Calculation of shortest-path betweenness: (a) When there is only a single shortest path from a source vertex s (top) to all other reachable vertices, those paths necessarily form a tree, which makes the calculation of the contribution to betweenness from this set of paths particularly simple, as describe in the text. (b) For cases in which there is more than one shortest path to some vertices, the calculation is more complex. First we must calculate the number of paths from the source to each other vertex (numbers on vertices), and then these are used to weight the path counts appropriately. In either case, we can check the results by confirming that the sum of the betweennesses of the edges connected to the source vertex is equal to the total number of reachable vertices — six in each of the cases illustrated here [43].*

3.2.2 Quantifying the strength of community structure of the algorithm

In order to know when the communities found by the algorithm are good ones, the same measure of the quality of a particular division of a network called *modularity*, which is also used by Louvain algorithm. If the number of within-community edges is no better than random, we will get $Q = 0$. Values approaching $Q = 1$, which is the maximum, indicate strong community structure. In practice, values for such networks typically fall in the range from about 0.3 to 0.7. Higher values are rare.

3.2.3 Limitations of the algorithm

The primary remaining difficulty with the algorithm is the relatively high computational demands it makes. The fastest of three measures, the one based on shortest-path betweenness, operates in $O(n^3)$ time for worst case on a sparse graph, which makes it usable for networks up to about 10000 vertices, but for larger systems it becomes intractable.

4 Experiments

4.1 Dataset

Files:

- **meta.dimacs10-polbooks** – Metadata about the network.
- **out.dimacs10-polbooks** – The adjacency matrix of the network in whitespace-separated values format, with one edge per line.

  The meaning of the columns in **out.dimacs10-polbooks** are:

  - **First column**: ID of from node
  - **Second column**: ID of to node

Data samples:

- 1, 2
- 1, 7
- 2, 4
- 2, 6
- 2, 7
- ...

As can be seen from the data samples, the line "1, 2" means that book 1 and book 2 are frequently co-purchased together.

4.1.1 Process the dataset

Processing the dataset includes 2 steps:

- Function **read-dimacs**: Read raw data from file and transfer to a list. Each element of the list is a list of 2 elements, each element is the value of each column
- Function **convert-to-graph**: Convert data to graph. We have to convert the data from **read-dimacs** output to **Graph** type from **network** package to use in **python-louvain** package as input, which has 2 steps. First, we get a list of non-duplicate nodes then add nodes to graph. Second, we add edges to graph, each edge is the value of each column.

```
procedure READ-DIMACS($path$) // path of the dataset
    Open and read the dataset with the $path$ then set to $f$
    Read all lines from $f$ then set to $lines$
    Initial the dataset array as $data$
    Loop each $line$ from $lines$
        Remove any leading and trailing whitespaces from $lines$ then split $line$ into a list
        Append $line$ to $data$
    return $data$
end procedure
```

```
procedure CONVERT-TO-GRAPH($data$) // data is the output of the READ-DIMACS function
    Initial the Graph object from networkx package as $G$
    Get a list of non-duplicate nodes from $data$ then set to $allMembers$
    Append $allMembers$ to $G$
    For each data point from $data$
        Add edge to $G$ from $data$
    return $G$
end procedure
```

4.2 Louvain algorithm

Python-louvain is a library written in python. The package name on pip is **python-louvain** but it is imported as **community** in python. More documentation for this module can be found at *http://python-louvain.readthedocs.io/*

4.2.1 Overview the code

The code has 3 main parts:

- Directory **dimacs10-polbooks** is where the dataset is saved.
- File **read-data.py** is where the dataset is processed.
- File **main.py** is where computes the best partition from the processed dataset and draw the graph.

Github: *https://github.com/luunguyenbaongoc/python-louvain/tree/master*

4.2.2 Compute and draw the graph

In this step, we just need to load the dataset from **read-data.py** module, then use the functions from **python-louvain** package to compute the best partition. Then, we draw the graph and color the nodes according to their partition.

```
procedure MAIN PROCESS
    Load dataset from CONVERT-TO-GRAPH function then set to $G$
    Compute the best partition from $G$ then set to $partition$
    Draw the graph from $G$ then set to $pos$
    Color the nodes according to their partition from $pos$
end procedure
```

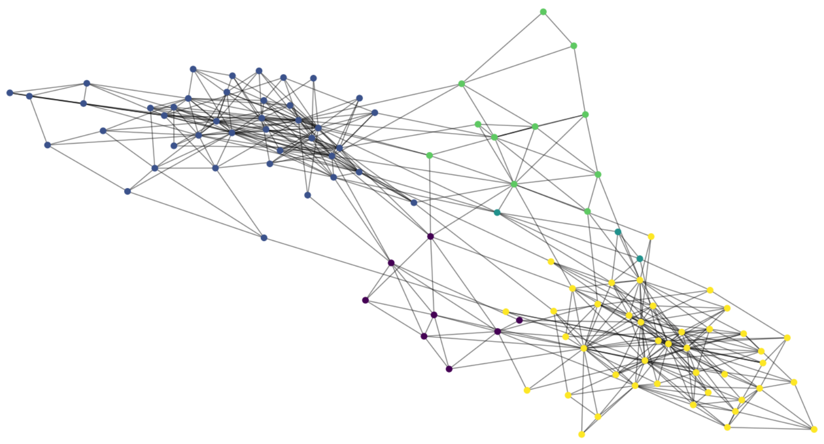

Figure 3: *Result of 5 communities in 0.144s with modularity of 0.526789*

4.3 Girvan-newman algorithm

This example shows the detection of communities using the **Girvan-Newman** method.

4.3.1 Overview the code

The code has 3 main parts:

- Directory **dimacs10-polbooks** is where the dataset is saved.
- File **read-data.py** is where the dataset is processed.
- File **main.py** is where computes the best partition from the processed dataset and draw the graph.

Github: *https://github.com/luunguyenbaongoc/girvan_newman/tree/master*

4.3.2 Compute and draw the graph

In this step, we just need to load the dataset from **read-data.py** module, then use the **girvan_newman** community function from **networkx** package to compute the best partition. Then, we draw the graph and color the nodes according to their partition.

```
procedure MAIN PROCESS
    Load dataset from CONVERT-TO-GRAPH function then set to $G$
    Compute the best partition from $G$ then set to $partition$
    Draw the graph from $G$ then set to $pos$
    Color the nodes according to their partition from $pos$
end procedure
```

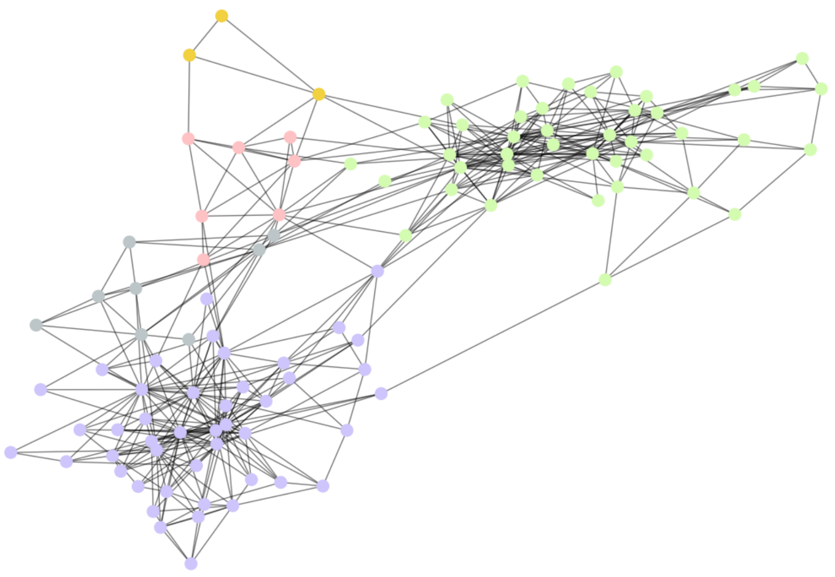

Figure 4: *Result of 5 communities in 0.425s with modularity of 0.516801*

4.4 Evaluation

The results of the Louvain algorithm are shown in the image. Vertices are colored according to the community they belong to. Overall, the clustering results of both Louvain and Girvan-Newman algorithms are good, even though there are few nodes visibly misplaced in communities. The algorithms are successful in identifying communities in the network and these communities appeared to be reasonable.

Both Louvain and Girvan-Newman algorithms have a degree of randomness, so the results might be different on different runs.

- **Number of communities:** The number of communities detected by the algorithms after different runs is mostly around 4-5. This number of communities seems reasonable, given the size and density of the network.
- **Modularity**: Most of the runs give value of modularity approximately around 0.53 and 0.52 for Louvain and Girvan-Newman respectively, which indicates a reasonably good community structure. This suggests that the both algorithms have found a community structure where there are significantly more edges within communities than between communities. However, based on the modularity scores, Louvain algorithm gives slightly better community structure than Girvan-Newman algorithm.
- **Community size:** Communities vary in size. Some runs give nearly equal size for all communities, suggesting that the topics covered by the books are evenly distributed. However, there are still some runs having some communities are much larger than others, it suggests that some topics are more popular or more interconnected than others.
- **Connections between communities:** Communities are connected to each other by a small number of edges. This suggests that the communities are relatively independent of each other.
- **Running time**: Based on the time complexity limitation of Girvan-Newman algorithm when the size of dataset increases, Louvain algorithm runs faster than Girvan-Newman algorithm as expected. For this case, most of the time Louvain is approximately 3 times faster than Girvan-Newman algorithm.

5 Conclusion

In conclusion, the Louvain algorithm emerges as a powerful tool for addressing the community detection problem within complex networks. Through its iterative and locally optimal approach, the algorithm efficiently identifies community structures that reveal underlying patterns of connectivity and organization within networks. While, the Newman-Girvan algorithm offers a valuable approach to community detection by leveraging the concept of edge betweenness centrality. It may be computationally intensive for large networks, its ability to identify communities based on the importance of edges in facilitating communication makes it a useful tool for understanding the structure and organization of complex networks.

One of the key strengths of the Louvain algorithm lies in its ability to handle large-scale networks with millions of nodes and edges, making it applicable to a wide range of real-world scenarios, including social networks, biological networks, and transportation networks. Its scalability is attributed to the greedy optimization process, which iteratively optimizes modularity, a measure that quantifies the strength of division of a network into communities. On the other hand, one notable strength of the Newman-Girvan algorithm is its effectiveness in identifying communities with well-defined boundaries, even in networks with complex structures and dense connections. By targeting edges with high betweenness centrality, the algorithm efficiently detects bridges connecting different communities, thereby revealing distinct community structures within the network.

However, it is important to acknowledge some limitations of the Louvain and Girvan-Newman algorithms. For Louvain algorithm, while it often produces high-quality community partitions, its reliance on local optimization may lead to sub-optimal solutions, particularly in networks with heterogeneous community sizes or hierarchical structures. Additionally, the algorithm's dependence on modularity as an objective function may result in the detection of communities that optimize modularity but do not necessarily correspond to meaningful or ground truth communities. Similarly, the Newman-Girvan algorithm is not without its limitations. Despite its effectiveness in detecting communities in many scenarios, the algorithm's reliance on edge betweenness centrality can pose challenges, especially in large-scale networks. Calculating betweenness centrality for all edges in the network can be computationally expensive, making the algorithm impractical for networks with millions of nodes and edges.

For the use case experiment **Customer network** with Political books dataset, based on different metrics, such as visualization, number of communities, community size, modularity, and connections between communities, we can see that both Louvain and Girvan-Newman algorithms succeed in detecting the communities and dividing the networks into different independent clusters with reasonable community numbers based on the visualization results. This result can be useful to understand the structure of the network and to identify types of customers based on their purchasing behavior.

References

[1] R. Campigotto, P. Conde Céspedes, and J.-L. Guillaume, “A generalized and adaptive method for community detection,” *arXiv preprint arXiv:1406.2518*, 2014.

[2] M. Rosvall and C. Bergstrom, “Maps of random walks on complex networks reveal community structure,” *Proc. Natl. Acad. Sci. USA*, vol. 105, pp. 1118–1123, 2008.

[3] R. Lambiotte, J.-C. Delvenne, and M. Barahona, “Laplacian dynamics and multiscale modular structure in networks.” arXiv 0812.1770, Dec 2008.

[4] T. Aynaud, V. D. Blondel, J.-L. Guillaume, and R. Lambiotte, *Multilevel Local Optimization of Modularity*, ch. 13, pp. 315–345. John Wiley & Sons, Ltd, 2013.

[5] D. Lui, K. Huang, C. Zhang, D. Wu, and S. Wu, “Study on discovery method of cooperative research team based on improved louvain algorithm,” *Scientific Programming*, 2021.

[6] J. Zhang, J. Fei, X. Song, and J. Feng, “An improved louvain algorithm for community detection,” *Mathematical Problems in Engineering*, 2021.

[7] C. Peng, T. G. Kolda, and A. Pinar, “Accelerating community detection by using k-core subgraphs,” 2014.

[8] E. A. Abbas and H. N. Nawaf, “Improving louvain algorithm by leveraging cliques for community detection,” in *2020 International Conference on Computer Science and Software Engineering (CSASE)*, pp. 244–248, 2020.

[9] Z. Zhou, W. Wang, and L. Wang, “Community detection based on an improved modularity,” in *Pattern Recognition* (C.-L. Liu, C. Zhang, and L. Wang, eds.), (Berlin, Heidelberg), pp. 638–645, Springer Berlin Heidelberg, 2012.

[10] S. Ryu and D. Kim, “Quick community detection of big graph data using modified louvain algorithm,” in *2016 IEEE 18th International Conference on High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, pp. 1442–1445, 2016.

[11] N. Ozaki, H. Tezuka, and M. Inaba, “A simple acceleration method for the louvain algorithm,” *International Journal of Computer and Electrical Engineering*, vol. 8, pp. 207–218, 2016.

[12] V. A. Traag, “Faster unfolding of communities: Speeding up the louvain algorithm,” *Phys. Rev. E*, vol. 92, p. 032801, Sep 2015.

[13] V. A. Traag, L. Waltman, and N. J. van Eck, “From louvain to leiden: guaranteeing well-connected communities,” *Scientific Reports*, vol. 9, p. 5233, Mar 2019.

[14] A. Aldabobo, A. Sharieh, and J. Riad, “An improved louvain algorithm based on node importance for community detection,” *Journal of Theoretical and Applied Information Technology*, vol. 100, no. 23, 2022.

[15] W. Du and X. He, “A common strategy to improve community detection performance based on the nodes’ property,” in *Bio-inspired Computing – Theories and Applications* (M. Gong, L. Pan, T. Song, and G. Zhang, eds.), (Singapore), pp. 355–361, Springer Singapore, 2016.

[16] O. Gach and J.-K. Hao, “Improving the louvain algorithm for community detection with modularity maximization,” in *Artificial Evolution: 11th International Conference, Evolution Artificielle, EA 2013, Bordeaux, France, October 21-23, 2013. Revised Selected Papers 11*, pp. 145–156, Springer, 2014.

[17] R. Rotta and A. Noack, “Multilevel local search algorithms for modularity clustering,” *ACM J. Exp. Algorithmics*, vol. 16, jul 2011.

[18] L. Waltman and N. J. van Eck, “A smart local moving algorithm for large-scale modularity-based community detection,” *The European Physical Journal B*, vol. 86, no. 471, 2013.

[19] M. Rosvall and C. T. Bergstrom, “Multilevel Compression of Random Walks on Networks Reveals Hierarchical Organization in Large Integrated Systems,” *PLoS ONE*, vol. 6, p. e18209, Apr. 2011.

[20] B. Yao, J. Zhu, P. Ma, K. Gao, and X. Ren, “A constrained louvain algorithm with a novel modularity,” *Applied Sciences*, vol. 13, no. 6, 2023.

[21] R. K. Darst, Z. Nussinov, and S. Fortunato, “Improving the performance of algorithms to find communities in networks,” *Phys. Rev. E*, vol. 89, p. 032809, Mar 2014.

[22] J. Reichardt and S. Bornholdt, “Statistical mechanics of community detection,” *Physical review E*, vol. 74, no. 1, p. 016110, 2006.

[23] P. C. Céspedes and J. F. Marcotorchino, “Comparing different modularization criteria using relational metric,” in *Geometric Science of Information* (F. Nielsen and F. Barbaresco, eds.), (Berlin, Heidelberg), pp. 180–187, Springer Berlin Heidelberg, 2013.

[24] L. Chaudhary and B. Singh, “Community detection using an enhanced louvain method in complex networks,” in *Distributed Computing and Internet Technology* (G. Fahrnberger, S. Gopinathan, and L. Parida, eds.), (Cham), pp. 243–250, Springer International Publishing, 2019.

[25] M. Schaub, J. Delvenne, R. Lambiotte, and M. Barahona, “Multiscale dynamical embeddings of complex networks,” *Physical Review E*, vol. 99, pp. 062308–1–062308–18, 2019.

[26] E. A. Leicht and M. E. Newman, “Community structure in directed networks,” *Physical review letters*, vol. 100, no. 11, p. 118703, 2008.

[27] L. Li, X. He, and G. Yan, “Improved louvain method for directed networks,” in *Intelligent Information Processing IX: 10th IFIP TC 12 International Conference, IIP 2018, Nanning, China, October 19-22, 2018, Proceedings 10*, pp. 192–203, Springer, 2018.

[28] P. Expert, T. S. Evans, V. D. Blondel, and R. Lambiotte, “Uncovering space-independent communities in spatial networks,” *Proceedings of the National Academy of Sciences*, vol. 108, no. 19, pp. 7663–7668, 2011.

[29] C. Wang, F. Wang, and T. Onega, “Network optimization approach to delineating health care service areas: Spatially constrained louvain and leiden algorithms,” *Transactions in GIS*, vol. 25, no. 2, pp. 1065–1081, 2021.

[30] R. Cazabet and G. Rossetti, “Challenges in community discovery on temporal networks,” *Temporal network theory*, pp. 181–197, 2019.

[31] N. Masuda and R. Lambiotte, *A guide to temporal networks.* World Scientific, 2016.

[32] V. Kawadia and S. Sreenivasan, “Sequential detection of temporal communities by estrangement confinement,” *Scientific reports*, vol. 2, no. 1, p. 794, 2012.

[33] D. Greene, D. Doyle, and P. Cunningham, “Tracking the evolution of communities in dynamic social networks,” in *2010 international conference on advances in social networks analysis and mining*, pp. 176–183, IEEE, 2010.

[34] P. J. Mucha, T. Richardson, K. Macon, M. A. Porter, and J.-P. Onnela, “Community structure in time-dependent, multiscale, and multiplex networks,” *science*, vol. 328, no. 5980, pp. 876–878, 2010.

[35] I. S. Jutla, L. G. Jeub, P. J. Mucha, *et al.*, “A generalized louvain method for community detection implemented in matlab,” *URL http://netwiki. amath. unc. edu/GenLouvain*, 2011.

[36] V. A. Traag and J. Bruggeman, “Community detection in networks with positive and negative links,” *Physical Review E*, vol. 80, no. 3, p. 036115, 2009.

[37] J. Pougué-Biyong and R. Lambiotte, “Louvain method for signed networks,” *In preparation.*

[38] P. Chunaev, “Community detection in node-attributed social networks: A survey,” *Computer Science Review*, vol. 37, p. 100286, 2020.

[39] T. Dang and E. Viennet, “Community detection based on structural and attribute similarities,” in *International Conference on digital society*, pp. 7–12, 2012.

[40] S. Citraro and G. Rossetti, “Identifying and exploiting homogeneous communities in labeled networks,” *Applied Network Science*, vol. 5, no. 1, p. 1–20, 2020.

[41] D. Combe, C. Largeron, M. Géry, and E. Egyed-Zsigmond, “I-Louvain: An Attributed Graph Clustering Method,” in *Intelligent Data Analysis*, (Saint-Etienne, France), LaHC, University of Saint-Etienne, France, Oct. 2015.

[42] H. Elhadi and G. Agam, “Structure and attributes community detection: Comparative analysis of composite, ensemble and selection methods,” in *Proceedings of the 7th Workshop on Social Network Mining and Analysis*, SNAKDD ’13, (New York, NY, USA), Association for Computing Machinery, 2013.

[43] M. E. Newman and M. Girvan, “Finding and evaluating community structure in networks,” *Physical review E*, vol. 69, no. 2, p. 026113, 2004.

[44] J. M. Anthonisse, *The rush in a directed graph.* Stichting Mathematisch Centrum. Mathematische Besliskunde, 1971.

[45] L. C. Freeman, “A set of measures of centrality based on betweenness,” *Sociometry*, pp. 35–41, 1977.

[46] G. R. Waissi, “Network flows: Theory, algorithms, and applications,” 1994.

[47] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms.* MIT press, 2022.

[48] M. E. Newman, “Scientific collaboration networks. ii. shortest paths, weighted networks, and centrality,” *Physical review E*, vol. 64, no. 1, p. 016132, 2001.

[49] U. Brandes, “A faster algorithm for betweenness centrality,” *Journal of mathematical sociology*, vol. 25, no. 2, pp. 163–177, 2001.