VIETNAM NATIONAL UNIVERSITY - HO CHI MINH CITY
HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

MATHEMATICAL FOUNDATION FOR COMPUTER SCIENCE

Assignment

Community Structure Identification: Book recomendation

Lecturer: **PhD. Nguyễn An Khương**

Students: **2270386 - Ngô Triệu Long**
2270376 - Tăng Quốc Thái
2270174 - Nguyễn Đình Khương
2210176 - Đỗ Võ Hoàng Hưng
2270375 - Nguyễn Trung Thuận

Ho Chi Minh city - November 2023

Contents

List of Figures

I. Introduction

1. Community Structure Identification Problem

The Community Structure Identification Problem (CSIP) is a foundational challenge in network analysis and graph theory. At its core, it grapples with the intricate task of unearthing the inherent groupings, or communities, within a network.

These communities hold the essence of dense internal connections, where nodes readily interact and share information among themselves. Yet, their outward tendrils reach less frequently, creating distinct boundaries with other gatherings within the network.

In a nutshell, the CSIP is about finding hidden groups or cliques within a network. It's like discovering friend groups within a school or social circles in a town.

These groups have tightly connected members but fewer ties to people outside their group. This problem is crucial in many fields because it helps us understand how complex systems are organized and how they function. Some example:

- **Social Networks:** Think about social networking sites like Facebook or Twitter. The challenge is to recognize unique clusters or communities of users who exhibit similar interests, share mutual social connections, or participate in comparable discussions. Identifying these communities enables the platform to understand user behavior better, tailor content suggestions, and identify influential users or potential communities of interest.

- **Biological Networks:** Analyzing communities in protein-protein interaction networks can help us understand cellular functions and identify potential drug targets. For instance, finding tightly connected groups of proteins involved in a disease process could lead to new therapies.

- **Online networks:** In online forums, discussion boards, or social media platforms, the Community Structure Identification Problem involves identifying clusters of users who interact and engage with each other on specific topics or interests. Uncovering these communities can help moderate online discussions, identify influential users, and facilitate content recommendation or targeted advertising.

Solving the Community Structure Identification Problem typically entails applying graph clustering algorithms to partition the network into cohesive com-

munities. These algorithms are techniques that place individuals into their most likely friend groups. They often try to maximize modularity, which measures how well-defined the communities are. It's like checking if the friend groups make sense or if people are randomly scattered.

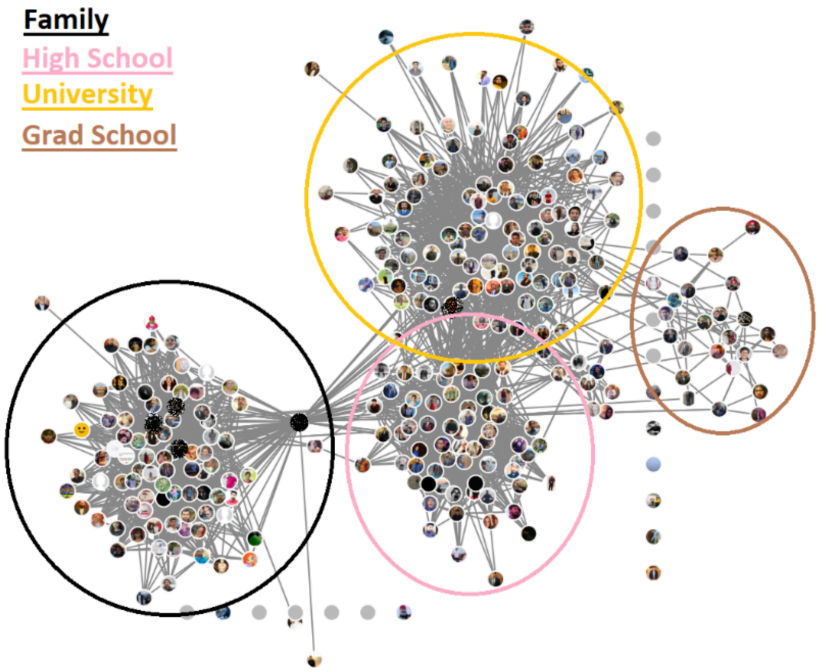

Figure 1: Figure 1.1 Social Network Example

2. CSIP for user cluster and product rating

2.1 Motivation

In the domain of E-commerce, characterized by a multitude of products vying for attention, the implementation of personalization emerges as a powerful strategy to transcend the noise. Imagine a digital storefront equipped with the ability to discern individual preferences even before user engagement, effectively directing customers towards products that evoke feelings of satisfaction and anticipation. This proposition is realized through the utilization of an application that harnesses the potential of user clustering and product rating prediction, thereby augmenting the overall shopping experience within this context.

By exploring the extensive collection of purchase histories and ratings, this technology uncovers interconnected communities of shoppers who share similar preferences and aspirations. It can be likened to mapping the lively virtual

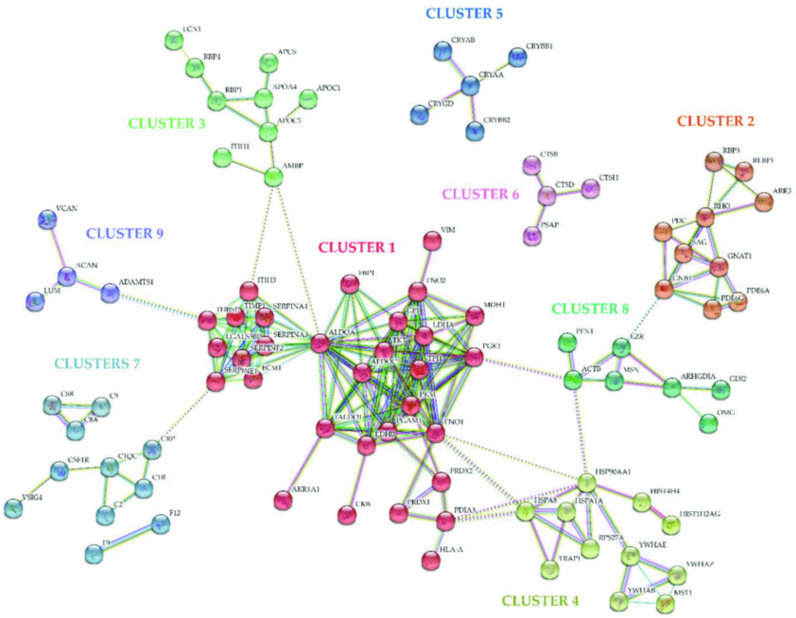

Figure 2: Figure 1.2 Protein Interaction Network example

marketplaces, revealing clusters where like-minded individuals gather around beloved brands, trending styles, or niche interests.

In this scenario, the Community Structure Identification Problem involves identifying clusters of users who share interests in product traits such as price, usability, maintainability, and brand,⋯and have bought similar products. By uncovering these user communities, we can gain insights into the organization of the buying trends and identify groups of users related to interests, purchase characteristics, or needs.

For businesses, this insight unlocks a treasure trove of opportunities:

- **Targeted Recommendations:** No more sifting through endless aisles. Each customer is greeted with a curated selection of products predicted to resonate with their unique preferences, fostering delight and satisfaction.
- **Personalized Discounts:**Promotions dance perfectly with individual desires, speaking to the heart of each cluster and igniting excitement for brands that truly understand their customers.
- **Strategic Product Development:** By analyzing the rating patterns of different clusters, businesses can uncover hidden gems, identify unmet needs,

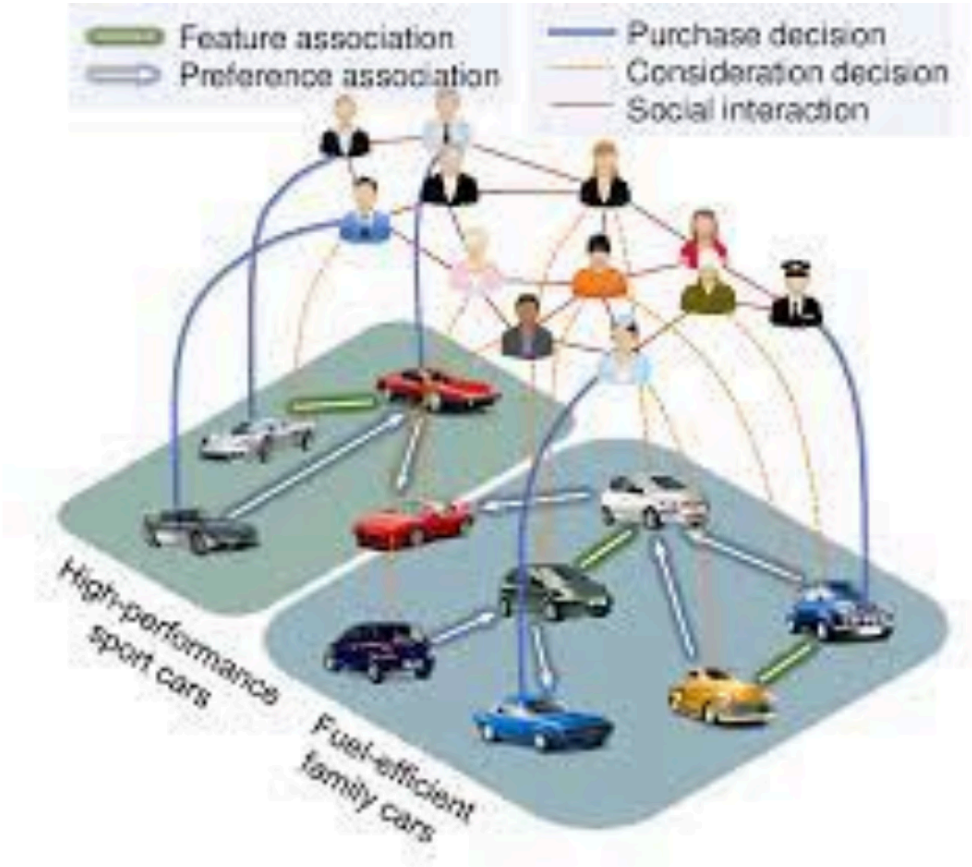

Figure 3: Figure 1.3 An example of how to cluster users based on product

and tailor their product offerings to align with the evolving desires of their customers.

- **Dynamic Pricing:** Prices gracefully adjust to reflect the anticipated value of a product within a specific cluster, optimizing revenue while ensuring a sense of fairness and value for shoppers.

Beside, the benefits would extend beyond immediate sales, shaping a more meaningful and engaging customer experience:

- **Relevant Content and Reviews:** Product descriptions and testimonials resonate more deeply when tailored to the interests of each cluster, fostering trust and connection.
- **Discovering New Delights:** Customers venture beyond their usual browsing habits, guided by recommendations that spark curiosity and introduce them to unexpected treasures they might have missed.
- **Building Brand Loyalty:** Personalized experiences cultivate a sense of belonging and appreciation, transforming customers into ardent advocates who eagerly share their positive experiences with others.

In summary, the Community Structure Identification Problem in the context of user clustering helps to reveal users with similar buying routines, combined with cluster average rating of a product to predict how a user who has never bought a product will rate it, enabling targeted marketing, recommendation systems, and a deeper understanding of market trend.

2.2 Dataset and approach proposal

Dataset Context: customer reviews and ratings of beauty-related products available on Amazon's platform.

Content: The dataset includes valuable information such as:

- Unique User IDs for customer identification.
- Product ASIN (Amazon's distinctive product identifier).
- Ratings, which reflect customer satisfaction on a scale from 1 to 5.
- Timestamps, recorded in UNIX time, indicate when the ratings were submitted.

This dataset is just a fragment of the extensive Amazon product dataset, encompassing a staggering 142.8 million reviews spanning May 1996 to July 2014. The complete dataset provides a wealth of information, including detailed product reviews, metadata, category information, pricing data, brand details, and image features.

Approach proposal

- Step 1: involves constructing a user-user bipartite graph. Edges connect users who share a purchase history for the same product, representing potential affinities between them. This network serves as the foundation for subsequent community detection algorithms.
- Step 2: deploys clustering algorithms to partition the network into cohesive communities. These techniques leverage edge density and topological features to identify groups of users exhibiting stronger internal connections than external linkages.
- Step 3: Step into the realm of predictive modeling. Given a product and a user who has never purchased it, we leverage the power of their assigned community. By aggregating the average rating for that product among the user's community members, we generate an expected rating reflecting the collective sentiment of their peers. This effectively leverages shared preferences within the identified communities to infer potential individual behavior.
- Step 4: concludes the process with rigorous evaluation. We compare the predicted ratings with actual ratings garnered once the user purchases the product. Metrics like mean squared error or absolute error assess the accuracy of our community-based predictions, allowing us to refine our algorithms and improve prediction performance.

In summary, users are divided into groups based on common characteristics, such as similarity of rating patterns or similar demographic data. After that, the prediction of the rating of a user to an item is computed based on the opinions of other users in the same group

CSI presents a compelling application of network analysis and community detection in customer behavior modeling. By uncovering hidden communities and harnessing the power of shared preferences, we can effectively predict user

ratings, personalize recommendations, and ultimately foster stronger customer relationships. This approach bridges the gap between individual interactions and collective behavior, revealing the hidden social fabric woven within large-scale online platforms.

II. Preliminaries

1. Networks and Node, Community Structure

A node is a unit of a network that represents an entity, such as a person, a group, or a concept. A network is a collection of nodes connected by edges or links, which can represent various types of relationships between the nodes. Networks are a common way of representing complex systems in a wide variety of scientific fields, from social sciences to biology, networks often exhibit a natural division into clusters or modules called communities. At the heart of these communities are nodes, which represent individual entities within the network. These nodes typically have unique attributes and connections to other nodes within the same community. A network is a collection of nodes, also called vertices, that are connected by edges or links, representing the relationships between them. The existence of community structure implies that vertices within a community share certain commonalities and exhibit similar behaviors within the larger graph.

For example, in a transportation network, nodes could represent cities or transportation hubs, with edges representing the links between them such as roads, railways, or airlines. Within this network, communities could emerge based on geographical proximity or shared transportation infrastructure. For example, cities located along the same highway or railway line may form a community, as they share similar transportation options and patterns of movement. Similarly, airlines may form communities based on shared routes or alliances, as they tend to have more connections and interactions with each other than with other airlines outside their community.

The term ”community structure” refers to the composition of a given network, including the number and attributes of nodes, as well as the relationships between them. Community structure is a crucial aspect of complex networks, as it indicates that nodes tend to cluster together into distinct groups or communities. Within each community, the density of edges is typically higher than between communities, emphasizing the strength of connections among members of the same community.

Understanding the community structure of a network can have important

implications for many real-world applications, such as predicting the spread of diseases or identifying influential individuals in a social network. Moreover, community detection can also help to reveal underlying patterns and structures in complex systems, providing insights into the organization and behavior of these systems.

2. Community Detection

Community detection is a fundamental problem in network analysis that aims to identify groups of nodes that are more similar to each other than to the rest of the network. Community detection is widely used in various fields such as biology, sociology, and marketing to understand social structures, reveal user data within the network, and develop relevant recommendation systems. There are several approaches to community detection, including hierarchical clustering, spectral clustering, modularity maximization, and random walk methods. Each method has its strengths and weaknesses, and the choice of method depends on the specific properties of the network being analyzed.

- Modularity maximization is one of the most widely used methods for community detection, and it aims to partition the network into non-overlapping communities that maximize a quality function called modularity. Modularity measures the density of edges within communities relative to the expected density if edges were distributed randomly. Maximal modularity indicates strong community structure, while low modularity suggests a random network.

- Hierarchical clustering is a method that groups nodes based on their similarity and generates a hierarchical tree of communities. In contrast, spectral clustering uses eigenvectors of the graph Laplacian to cluster nodes and has been shown to perform well on networks with well-defined community structures.

- The Girvan-Newman algorithm is a popular method for community detection in networks. The algorithm works by iteratively removing edges from the network, with the aim of breaking it into smaller components that correspond to communities.

Community detection has many applications, including identifying functional modules in biological networks, understanding the structure of social networks,

and detecting communities in web graphs. In recent years, there has been growing interest in developing community detection algorithms that are scalable, efficient, and applicable to large-scale networks.

3. Modularity

A good community partition of a network should have fewer edges between communities than one would expect by chance. This indicates significant community structure within the network. On the other hand, if the number of edges between groups is what would be expected by chance, it is not evidence of significant community structure. The measure known as modularity quantifies this idea that true community structure corresponds to a statistically surprising arrangement of edges, with fewer edges between groups and more edges within groups than expected by chance.

3.1 Modularity Density

Li et al. have developed a new quantitative metric called modularity density (D), which is based on the density of subgraphs, to determine community structure of networks.
The higher the value of D, the better is a partition. The optimization of modularity density is also NP-hard. The modularity density is defined as:

$$D_\lambda = \sum_{i=1}^{m} \frac{2\lambda L(V_i, V_i) - 2(1-\lambda)L(V_i - \overline{V1})}{|V_i|}$$

where, V_i is the subset of V i=1,..., m, such that $L(V_i) = \sum_{i \in V_i, j \in \overline{V_1}} A_{ij}$, where $\overline{V_1} = VV_i$

3.2 Modularity Q

Modularity Q is one of the most used measure. The modularity Q is, up to a multiplicative constant, the number of edges falling within groups minus the expected number in an equivalent network with edges placed at random. The modularity can be either positive or negative, with positive values indicating the possible presence of community structure. Thus, one can search for community structure precisely by looking for the divisions of a network that have positive, and preferably large, values of the modularity.
The original idea of modularity was given by Newman and Girvan, they have

defined modularity Q as:

$$Q = \frac{1}{2}\sum_{ij}(A_{ij} - \frac{k_i k_j}{2m})\delta(\sigma_i, \sigma j)$$

Here, m is the number of links, k_i is the degree of vertex i, k_j is the degree of vertex j, σ_i is the community to vertex i, σ_j is the community to vertex j, and $\delta(\sigma_i \sigma_j) = 1$ if i and j belong to the same community, otherwise it equals to 0.
An alternate formulation of this is as a sum over communities:

$$Q = \frac{1}{2m}\sum_{c}(A_c - \frac{K_c^2}{4m})$$

where m_c is the number of internal edges (or tatal internal edge weight) of community c and $K_c = \sum_{i|\sigma_i=c} k_i$ is the total (weighted) degree of nodes in community c.

III. Approaches

1. Community Detection Methods in Network Analysis: Agglomerative and Divisive Approaches

Community analytics, an essential task in network analysis, involves the identification of groups or communities within a network structure. Two primary types of methods are commonly employed for community detection: agglomerative methods and divisive methods.

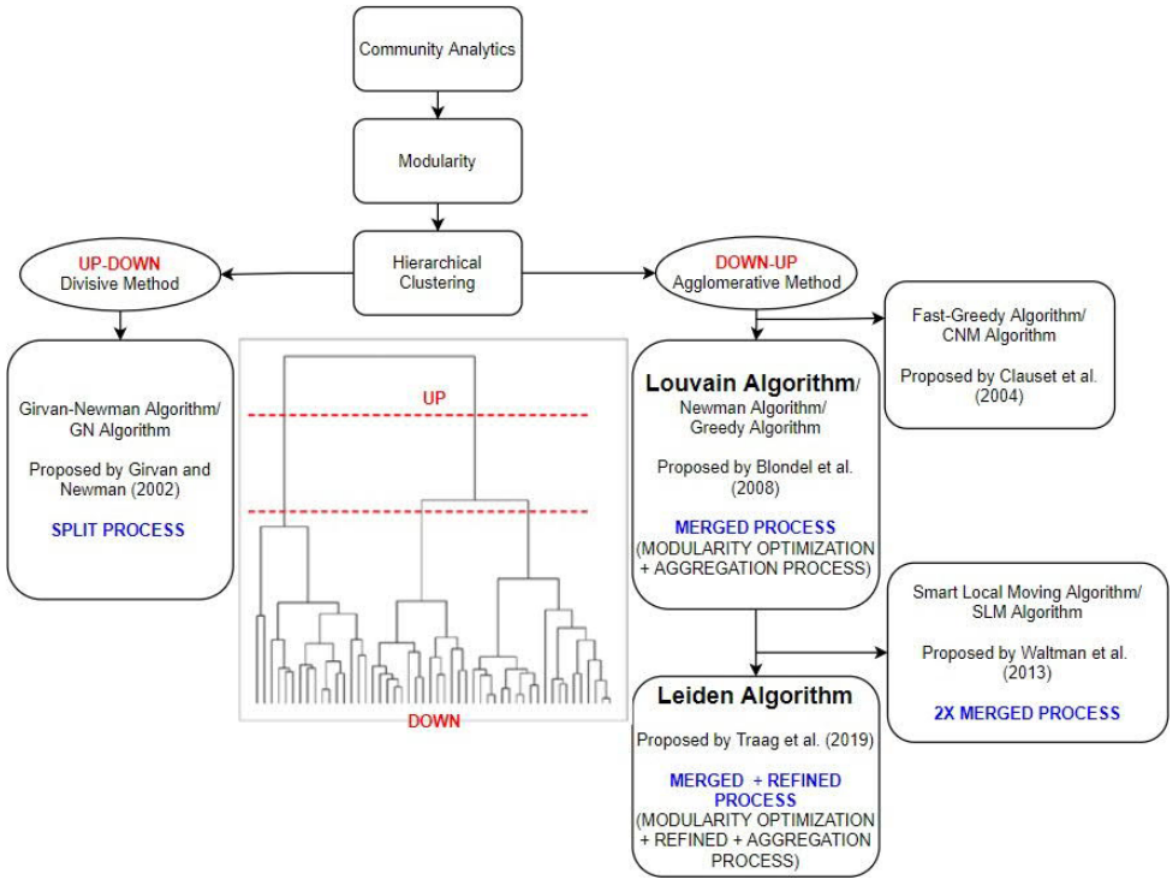

Figure 1. Taxonomy of community analytics.

In agglomerative methods (illustrated on the right side of Figure 1), the process initiates with individual nodes and gradually merges them based on their similarity or connectivity, leading to hierarchical clustering.

This approach merges nodes from the bottom up, progressively forming larger communities. The agglomerative category includes algorithms such as Louvain and Leiden, which utilize modularity and hierarchical clustering.

Modularity, represented as Q, measures community strength and guides the algorithm. Higher modularity values indicate better communities, while a value below 1 suggests that each node is treated as a separate community.

The dendrogram in Figure 1 represents the hierarchy of clusters generated by hierarchical clustering. On the other hand, divisive methods (depicted on the left side of Figure 1) start with a single partition containing all nodes and iteratively split it by removing edges with low similarity.

This process, known as the split process, aims to find smaller, more cohesive communities by progressively dividing the network. Although agglomerative and divisive methods employ different approaches, both ultimately aim to unveil meaningful groups of nodes based on their connectivity patterns within the network.

2. Girvan-Newman algorithm

The Girvan-Newman algorithm is a well-known community detection algorithm in network analysis. Proposed by Michelle Girvan and Mark Newman, it aims to identify communities or clusters within a network by iteratively removing edges with high betweenness centrality. This algorithm follows a divisive approach, progressively dividing the network into smaller communities based on the importance of edges.

Edge Betweenness: It uses edge betweenness to find and remove central edges that connect communities within a larger graph. The formula for edge betweenness is:

$$B(e) = \sum_{u,v \in V(G)} \frac{\sigma_{u,v}(e)}{\sigma_{u,v}} \tag{1}$$

where σ_u, v is the number of shortest paths between two distinct vertices and $\sigma_{u,v}(e)$ is the corresponding number of shortest paths containing a particular edge.

Modularity: After removing an edge, the Girvan-Newman algorithm calculates the modularity (Q) of the graph, which is a value between the range $[-0.5, 1]$. A higher value suggests a more significant community structure. Therefore, we can identify communities by maximizing modularity.

The formula for modularity is:

$$Q = \sum_{s=1} m \left[\frac{l_s}{L} - \left(\frac{d_s}{2L} \right)^2 \right] \tag{2}$$

(m = number of modules, ls = the number of edges inside module s, L = the number of edges in the network, d = total degree of the nodes in module s)

This process of removing an edge and calculating the modularity is iteratively repeated. The algorithm will stop when the new modularity is no longer greater than the modularity from the previous iteration. The ending modularity is usually around 0.6.

2.1 Pseudo Code

Algorithm 1 Girvan-Newman Algorithm

Input : Graph G

Output: Disconnected components of G

do
  Let n be the number of edges in G
  for $i = 0$ to $n - 1$ **do**
    Let $B[i]$ be betweenness centrality of edge i
    if $B[i] > max_B$ **then**
      $max_B \leftarrow B[i]$
      $max_{B(edge)} \leftarrow i$
    end
    Remove edge i from graph
  end
while number of edges in G is not 0;

2.2 Implement Code

```
def girvan_newman_algorithm(G):
    # calculate initial betweenness centrality of all edges
    betweenness = nx.edge_betweenness_centrality(G)

    while len(G.edges()) > 0:
        # remove edge with highest betweenness centrality
        max_edge = max(betweenness, key=betweenness.get)
        G.remove_edge(*max_edge)

        # recalculate betweenness centrality of remaining edges
        betweenness = nx.edge_betweenness_centrality(G)

        # check if the graph has been split into disconnected
    components
```

```
14          if nx.number_connected_components(G) > 12:
15              # if so, return the list of communities
16              communities = list(nx.connected_components(G))
17              return communities
18
19      # if no disconnected components are found, return the original
        graph as a single community
20      return [set(G.nodes())]
```

2.3 Pros, Cons

The Girvan-Newman algorithm offers several advantages and disadvantages in community detection within network analysis.

On the positive side, the algorithm is effective in uncovering the community structure of a network. By targeting edges with high betweenness centrality and iteratively removing them, it reveals the underlying modular organization of the network. This capability allows for a comprehensive understanding of how communities are formed and interconnected.

Additionally, the Girvan-Newman algorithm does not require any prior information or assumptions about the network's structure or the number of communities. It dynamically adapts to the unique characteristics of the network during the divisive process, making it versatile and applicable in various scenarios.

The algorithm also provides valuable insights into bridge edges, which are edges with high betweenness centrality that act as connections between different communities. By identifying these bridge edges, researchers can gain a deeper understanding of the interconnections and relationships between communities.

However, there are also some drawbacks to consider. One major limitation is the computational expense associated with the Girvan-Newman algorithm, especially for large networks. Calculating betweenness centrality for all edges in the network requires significant computational resources, making it impractical for networks with millions of nodes.

Another limitation is the resolution limit of the algorithm. It may struggle to detect smaller communities or communities with weaker connectivity. In some cases, the algorithm tends to merge smaller communities into larger ones, resulting in the loss of fine-grained community structure.

Additionally, the Girvan-Newman algorithm lacks scalability for very large networks. As the network size increases, the algorithm's runtime and memory

requirements grow significantly, posing challenges in analyzing massive datasets.

Despite these limitations, the Girvan-Newman algorithm remains a valuable tool for community detection, providing insights into the modular organization of networks. Researchers continue to explore enhancements and alternative approaches to address its drawbacks and improve its applicability in different network analysis scenarios.

2.4 Time Complexity

Despite Girvan-Newman' s popularity and quality of community detection, it has a high time complexity, increasing up to $O(m^2n)$ on a sparse graph having m edges and n nodes. As a result, Girvan-Newman is generally not used on large scale networks. Its optimal node count is a few thousand nodes or less.

Because of this, there exist greedy algorithms for detecting communities to reduce the time but at the same time sacrificing the most accurate results. One such example is the Louvain algorithm.

3. Louvain Algorithm

The Louvain algorithm is a fast implementation of community detection. It can be used to analyze a network of 2 million nodes in only 2 minutes

It is a hierarchical clustering algorithm that involves two phases: modularity optimization and community aggregation

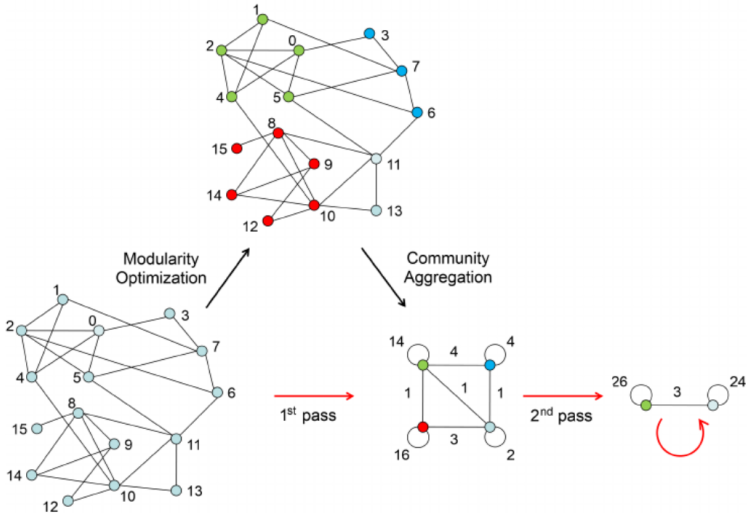

Figure 4: Louvain Algorithm

3.1 Modularity Optimization

As seen in the figure, the first step is to optimize the modularity of the entire graph. In this example, it splits the nodes into four communities.

To find these clusters, each node is moved into its neighboring community. If the change in modularity (ΔQ) is greater than 0, it is moved into the neighboring community, Otherwise, it remains in its current community. This process is repeated until $\Delta Q = 0$ for all nodes.

3.2 Community Aggregation

After modularity optimization, super nodes are created to represent each cluster. After the initial phase of the algorithm, there will exist many communities.

However, the two phases repeat, creating larger and larger communities. The algorithm stops only when no improvement can be made by any of the two operations

3.3 Pros, Cons

Pros of the Louvain algorithm:

- Fast and scalable for large networks.
- Optimizes modularity, which measures community quality.
- Detects hierarchical community structures.
- Allows flexibility in resolution for different levels of detail.
- Widely used with extensive resources and documentation.

Cons of the Louvain algorithm:

- Resolution limit may merge small communities into larger ones.
- Results can be sensitive to initial community assignments.
- Assumes non-overlapping communities.
- Bias towards detecting large, cohesive communities.
- Lacks theoretical guarantees of optimality.

We need to keep in mind these points when considering the Louvain algorithm for community detection.

3.4 Pseudo Code

Algorithm 2 Louvain Algorithm

Input : Graph G

Output: Communities

Initialize each node in G as a separate community
Initialize the change indicator $modularity_{improvement}$
repeat
    modularity_improvement $\leftarrow 0$
    for each node i in G **do**
        for each neighbor j of node i **do**
            Merge community of node i with community of node j
            Compute the modularity improvement
            Undo the merge if the improvement is not positive
            Track the maximum improvement
        end
    end
until modularity_improvement is not positive;

3.5 Implement Code

```
1
2  def initialize_communities(graph):
3      """
4      Initialize each node into its own community.
5      """
6      return {node: [node] for node in graph.nodes()}
7
8
9  def calculate_modularity(graph, communities, m):
10     """
11     Calculate the modularity of the graph given the communities.
12     """
13     Q = 0.0
14     for community in communities.values():
15         community_subgraph = graph.subgraph(community)
16         L_c = sum(dict(community_subgraph.degree(weight='weight')).
       values())
17         d_c = 2 * L_c / m
```

```
        Q += (L_c / m) - np.square(d_c)
    return Q


def louvain(graph):
    """
    Perform community detection using the Louvain algorithm.
    """
    m = sum(dict(graph.degree(weight='weight')).values()) / 2

    # Step 1: Create initial communities
    communities = initialize_communities(graph)

    # Track the best modularity
    best_modularity = calculate_modularity(graph, communities, m)

    while True:
        # Step 2: Move nodes between communities
        moved = False

        for node in graph.nodes():
            # Get the current community of the node
            current_community = None
            for c, nodes in communities.items():
                if node in nodes:
                    current_community = c
                    break

            # Calculate the modularity gain of moving the node to each neighboring community
            neighbor_communities = set()
            for neighbor in graph.neighbors(node):
                for c, nodes in communities.items():
                    if neighbor in nodes:
                        neighbor_communities.add(c)

            modularity_gains = {}
            for neighbor_community in neighbor_communities:
                communities[current_community].remove(node)
                communities[neighbor_community].append(node)

                modularity_gains[neighbor_community] = calculate_modularity(graph, communities, m)
```

```
59
60                  # Move the node to the community with the maximum
    modularity gain
61                  max_gain_community = max(modularity_gains, key=
    modularity_gains.get)
62                  max_gain = modularity_gains[max_gain_community]
63
64                  if max_gain > 0:
65                      communities[max_gain_community].append(node)
66                      moved = True
67                      break
68
69                  # If no improvement, revert the move
70                  communities[neighbor_community].remove(node)
71                  communities[current_community].append(node)
72
73          # If no nodes have moved communities, stop iterating
74          if not moved:
75              break
76
77          # Recalculate modularity
78          modularity = calculate_modularity(graph, communities, m)
79
80          # If modularity has improved, update the best modularity
81          if modularity > best_modularity:
82              best_modularity = modularity
83
84      return communities.values()
```

4. Leiden algorithm

The Leiden algorithm is an algorithm for detecting communities in large networks. It separates nodes into disjoint communities so as to maximize a modularity score for each community. Modularity quantifies the quality of an assignment of nodes to communities, that is how densely connected nodes in a community are, compared to how connected they would be in a random network. The Leiden algorithm extends the Louvain algorithm, which is widely seen as one of the best algorithms for detecting communities. However, the Louvain algorithm can lead to arbitrarily badly connected communities, whereas the Leiden algorithm guarantees communities are well-connected.

This algorithm is processed included 3 step:

- Step 1: local moving of nodes.
- Step 2: Refinement of the partition.
- Step 3: Aggregation of the network based on the refined partition.

We can see this process in the picture from (a) to (f) as below. It starts from a singleton partition (a). The algorithm moves individual nodes from one community to another to find a partition (b), which is then refined (c). An aggregate network (d) is created based on the refined partition, using the non-refined partition to create an initial partition for the aggregate network. For example, the red community in (b) is refined into two subcommunities in (c), which after aggregation become two separate nodes in (d), both belonging to the same community. The algorithm then moves individual nodes in the aggregate network (e). In this case, refinement does not change the partition (f). These steps are repeated until no further improvements can be made.

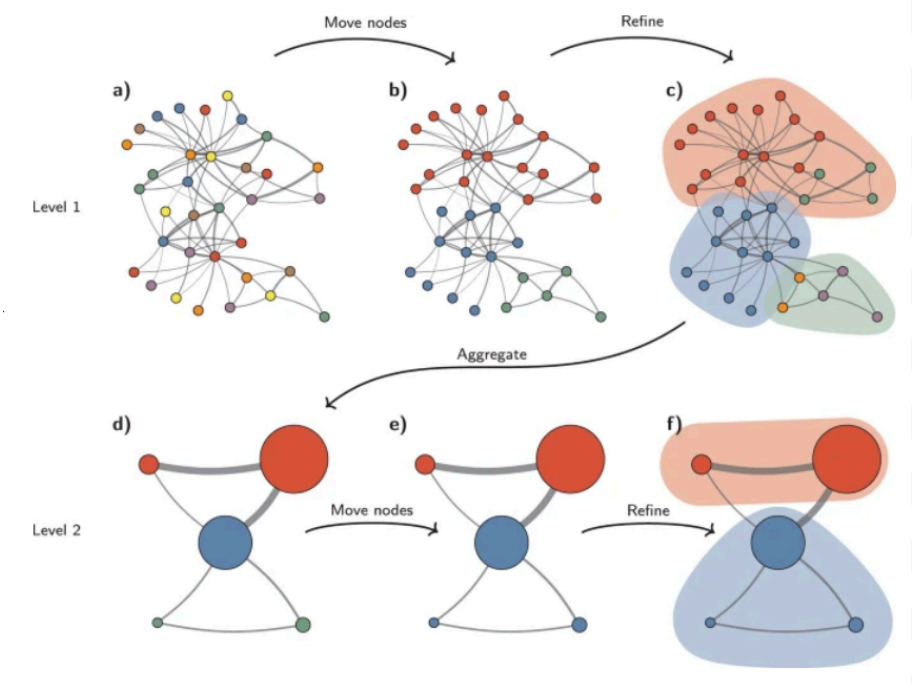

Figure 5: The Leiden algorithm

4.1 Pseudo code

Algorithm 3.2: Leiden

Input : Graph G, Partition P

function Leiden(Graph G, Partition P)
  do
    $P \leftarrow$ FastLocalMoving(G, P)
    $done \leftarrow |P| == |V(G)|$ `// Every community consists of one node`
    if not *done* **then**
      $P_{refined} \leftarrow$RefinePartition(G, P)
      $G \leftarrow$Coarsen$(G, P_{refined})$
      $P \leftarrow$SingletonPartition(G)
  while not *done*
  return flatten*(P)*

function FastLovalMoving(Graph G, Partition P)
  $Q \leftarrow$ Queue$(V(G))$ `// Insert nodes in random order`
  do
    $v \leftarrow Q$.pop()
    $C_{new} \leftarrow$argmax$_{C \in P \cup \emptyset}$ $\Delta_{v \to C}$
    if $\Delta_{v \to C_{new}} > 0$ **then**
      $v \to C_{new}$ `// Move` v `to Community` C_{new}
      $Neighbours \leftarrow \{u \mid \{v, u\} \in E(G), u \notin C_{new}\}$
      Q.push$(Neighbours \setminus Q)$
  while $Q \neq \emptyset$
  return P

function RefinePartition(Graph G, Partition P)
  $P_{refined} \leftarrow$ SingletonPartition(G)
  for $C \in P$ **do**
    $P_{refined} \leftarrow$ MergeNodesSubset$(G, P_{refined}, C)$
  return $P_{refined}$

function MergeNodesSubset*(Graph G, Partition P, Subset S)*
  $R \leftarrow \{v \mid v \in S, \text{cut}_w(v, S - v) \geq \deg_w(v)[\text{vol}_w(S) - \deg_w(v)]\}$
  for $v \in R$ **do**
    if v is singleton **then**
      $\mathcal{T} \leftarrow \{C \mid C \in P, C \subseteq S, \text{cut}_w(C, S - C) \geq \text{vol}_w(C)[\text{vol}_w(S) - \text{vol}_w(C)]\}$
      $C_{new} \leftarrow$argmax$_{C \in \mathcal{T}}$ $\Delta_{v \to C}$
      $v \to C_{new}$

4.2 Pros, Cons

The Leiden algorithm is a popular community detection algorithm that has been used in various fields, including social network analysis and bioinformatics. Some of the advantages of the Leiden algorithm are:

- It is a fast and scalable algorithm that can handle large and complex networks.
- It can detect communities of different sizes and densities.

- It can handle directed and weighted networks.
- It has been shown to perform well in comparison to other community detection algorithms.

However, there are also some potential limitations and drawbacks of the Leiden algorithm, such as:

- It may not be suitable for detecting communities with overlapping nodes.
- It may not perform as well in networks with low modularity.
- It may be sensitive to the choice of resolution parameter.
- It may require some expertise in parameter tuning and interpretation of results.

IV. Experiments

1. Preprocessing

The dataset is quite big to handle contain 3 file:

- Books.csv: 271379 records
- Users.csv: 276271 records
- **Ratings.csv: 1149781 records**

Therefore, we are using the map-reduce technique to split the data in order to fit the computer memory.

The map reduce is using to group and count all pair of books bought by at least 2 users and running on file Ratings.csv.

Map reduce first phase:

Map phase	**Reduce phase**
<User-ID1, ISBN1> <User-ID1, ISBN2> <User-ID1, ISBN3>	<User-ID1, [ISBN1, ISBN2, ISBN3, ...]>

Table 1: Map reduce first phase find all books were bought by single users

Map second phase:

Map phase	
<User-ID1, [ISBN1, ISBN2, ISBN3, ...]>	<[ISBN1, ISBN2], User-ID1> <[ISBN1, ISBN3], User-ID1> <[ISBN2, ISBN3], User-ID1>

Table 2: Map phase to find all books pair of book bought by users

Reduce second phase:

Reduce phase	
<[ISBN1, ISBN2], User-ID1>	<[ISBN1, ISBN2], Count1>
<[ISBN1, ISBN3], User-ID1>	<[ISBN1, ISBN3], Count2>
<[ISBN2, ISBN3], User-ID1>	<[ISBN2, ISBN3], Count3>

Table 3: Reduce phase to find all books pair of books and number of users bought it

2. Build application

We built a web application to demonstrate the process of recommendation in book store. Web are built with VueJS as a Frontend and Flask as a framework by Python language programming. Backend is in charge of query similar books and return it to Frontend via Restful api.

Figure 6: Technologies used

2.1 Build Backend

Flask introduction: Flask is a micro web framework written in Python. It is classified as a microframework because it does not require particular tools or libraries. It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions.

All required libraries:

Create file requirements.txt:

```
Flask==2.2.3
Flask-Cors==3.0.10
pandas==2.0.0
networkx==2.8.8
```

Install all libraries via command:

```
pip install -r requirements.txt
```

Run backend locally on port 5000

```
python main.py
```

2.2 Build Frontend

VueJS introduction: Vue.js is an open-source model–view–viewmodel front end JavaScript framework for building user interfaces and single-page applica-

tions. It was created by Evan You, and is maintained by him and the rest of the active core team members

2.2.1 Components

Include 3 pages

1. Homepage: Some categories, and popular items among users

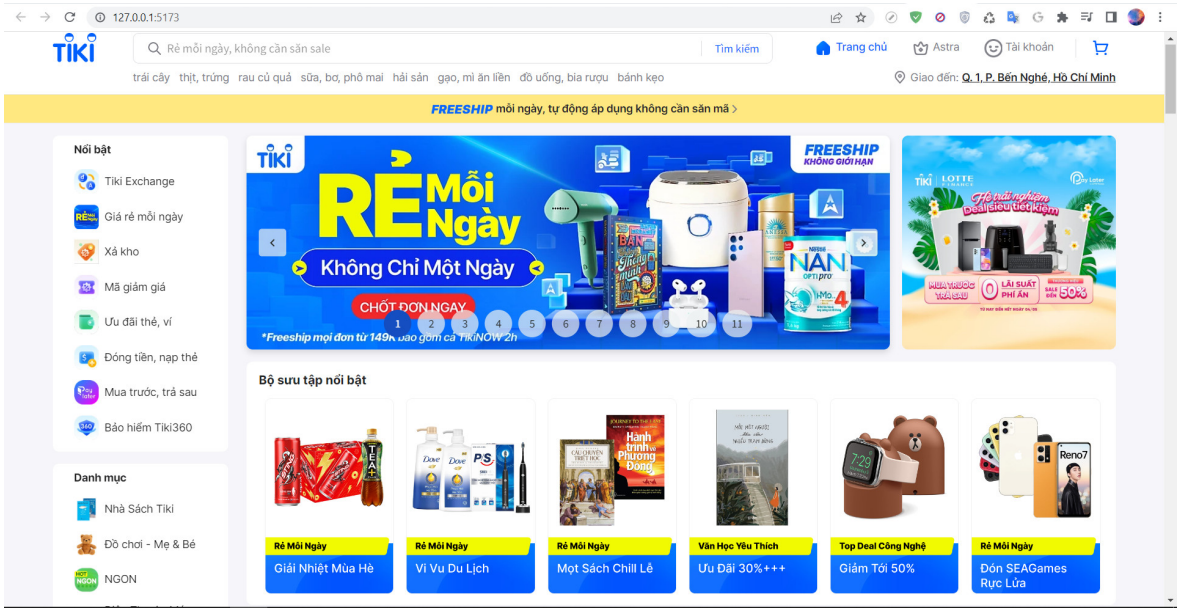

Figure 7: Homepage

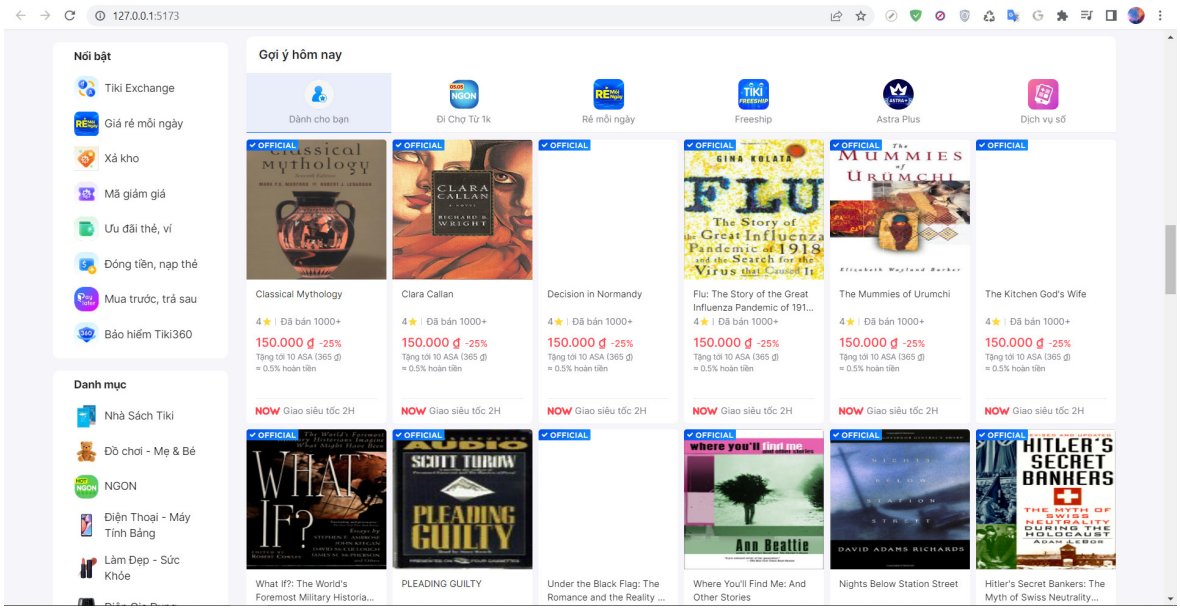

Figure 8: Book category in homepage

2. Book category page: All books in E-Commerce using pagination.

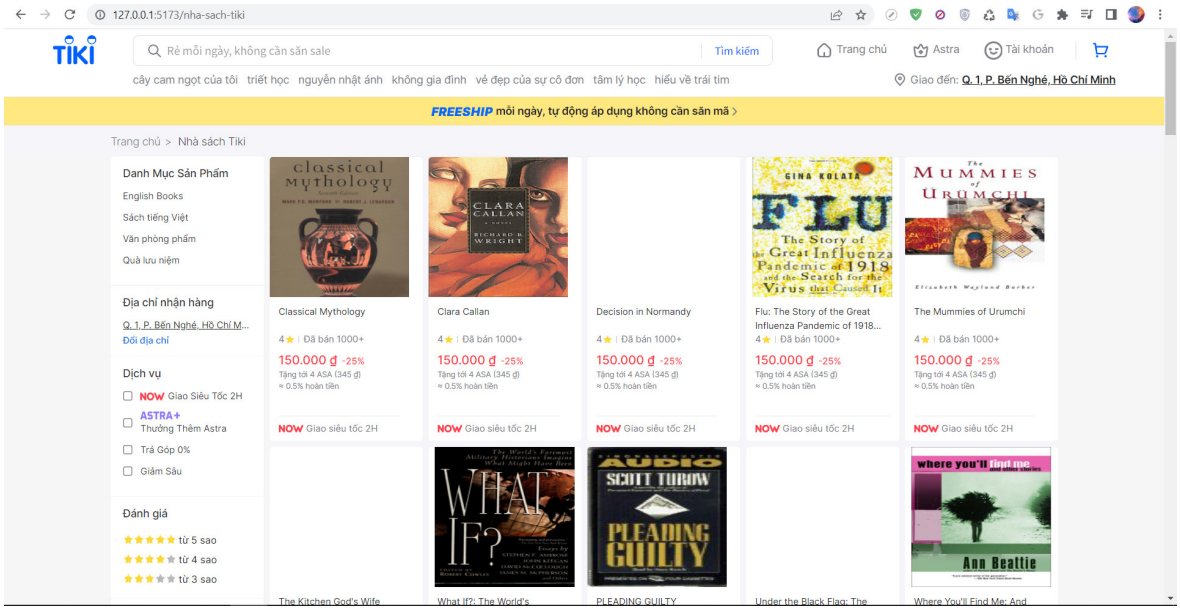

Figure 9: Book category page

3. Book detail page: Detail of a book and all similar book using Louvain, Leiden, Girvan Newman.

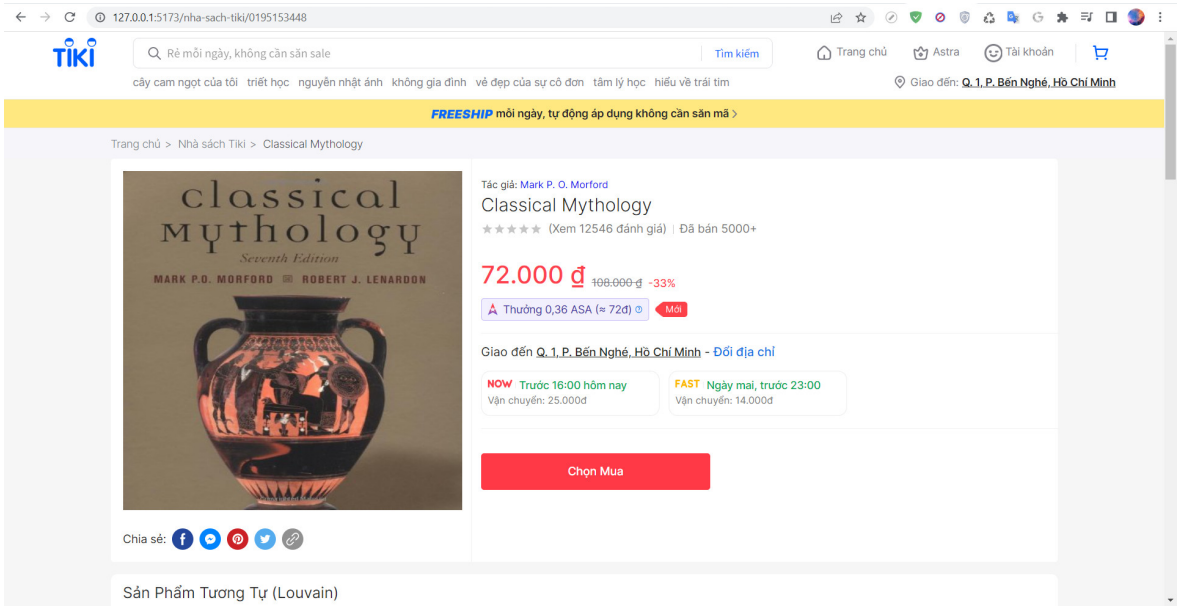

Figure 10: Book detail page

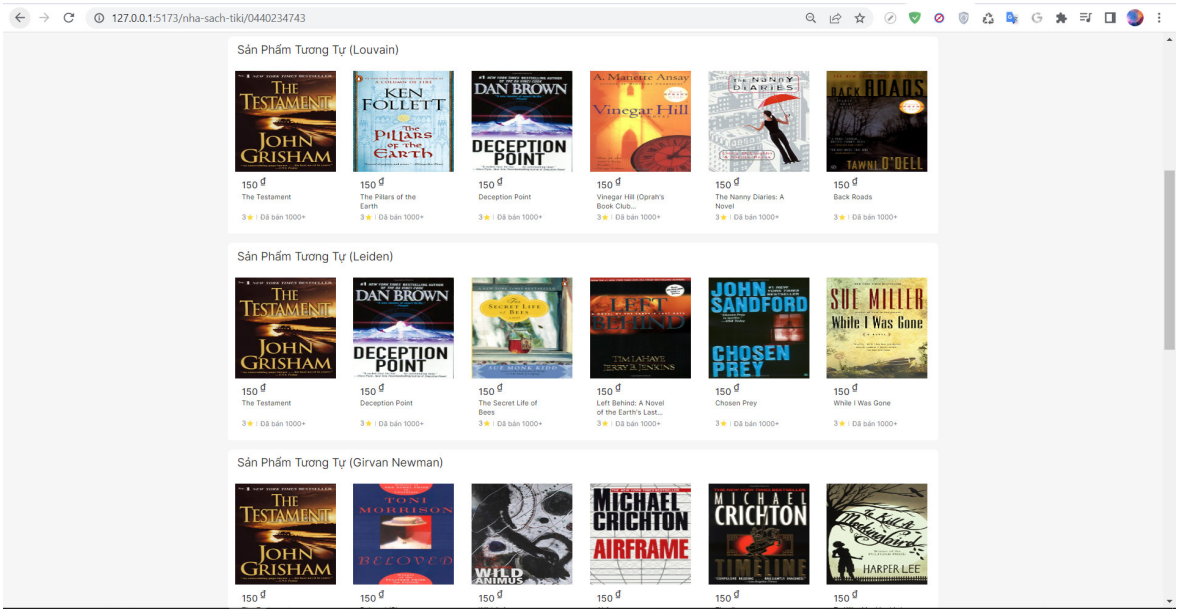

Figure 11: Book detail page recommendation algorithms

3. Modularity score comparison

To improve the efficiency of measuring algorithm modularity, we implemented a strategy of dividing the dataset into smaller subsets. Our approach involved selecting the top 500 books that appeared in all book pairs from the previous Map Reduce step. This selection criterion was chosen over random selection from all pairs to avoid generating a sparse graph that would result in significantly lower modularity scores for all three algorithms, making it challenging to measure them accurately.

By ensuring that the selected nodes had the highest number of edges, we aimed to facilitate faster execution of all three algorithms

```
# print number of edges and nodes
print("Number of edges:", G.number_of_edges())
print("Number of nodes:", G.number_of_nodes())
```

```
Number of edges: 15652
Number of nodes: 500
```

Figure 12: Number of edge and node for modularity measurement

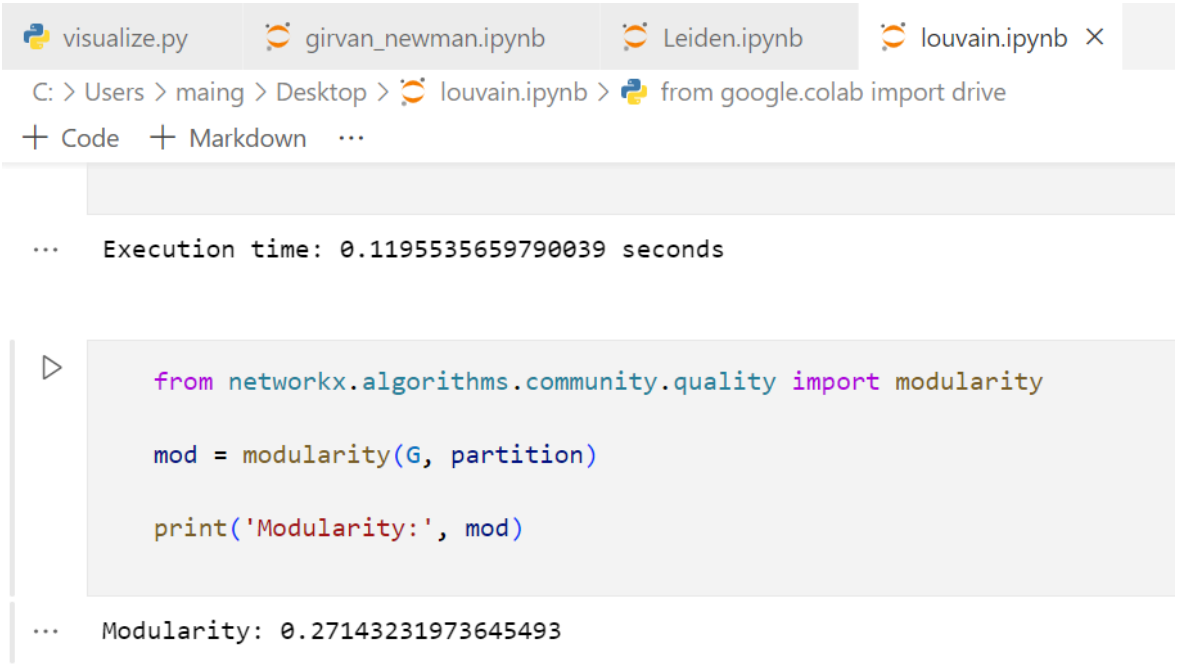

Figure 13: modularity measurement for Louvain algorithms

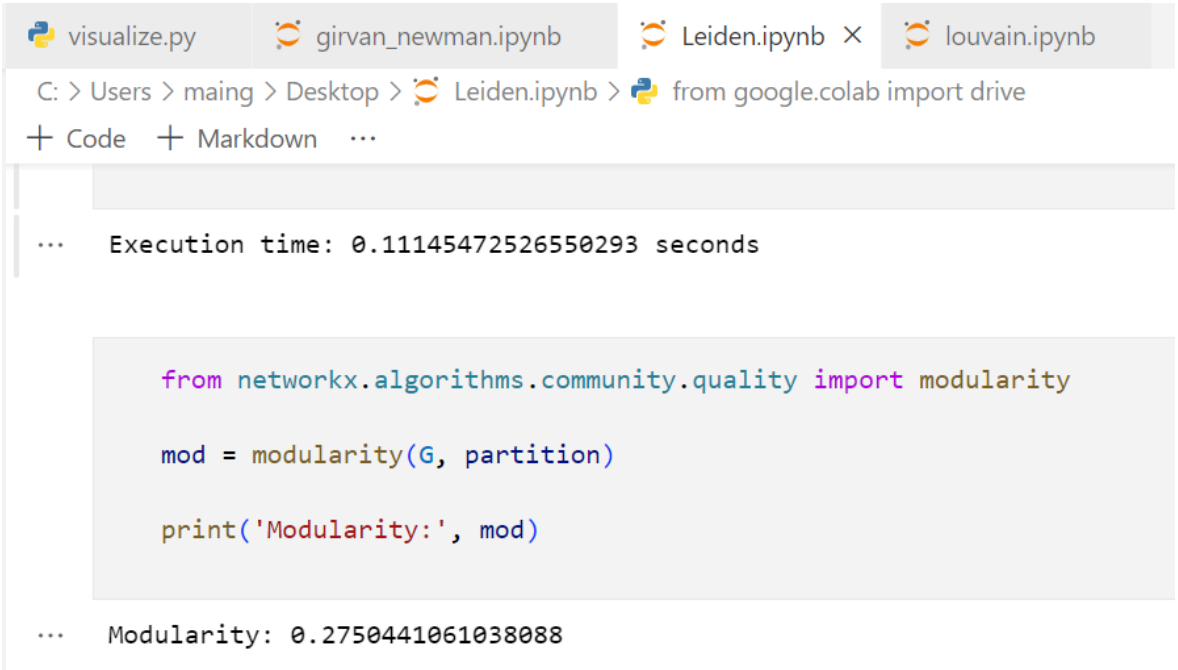

Figure 14: modularity measurement for Leiden algorithms

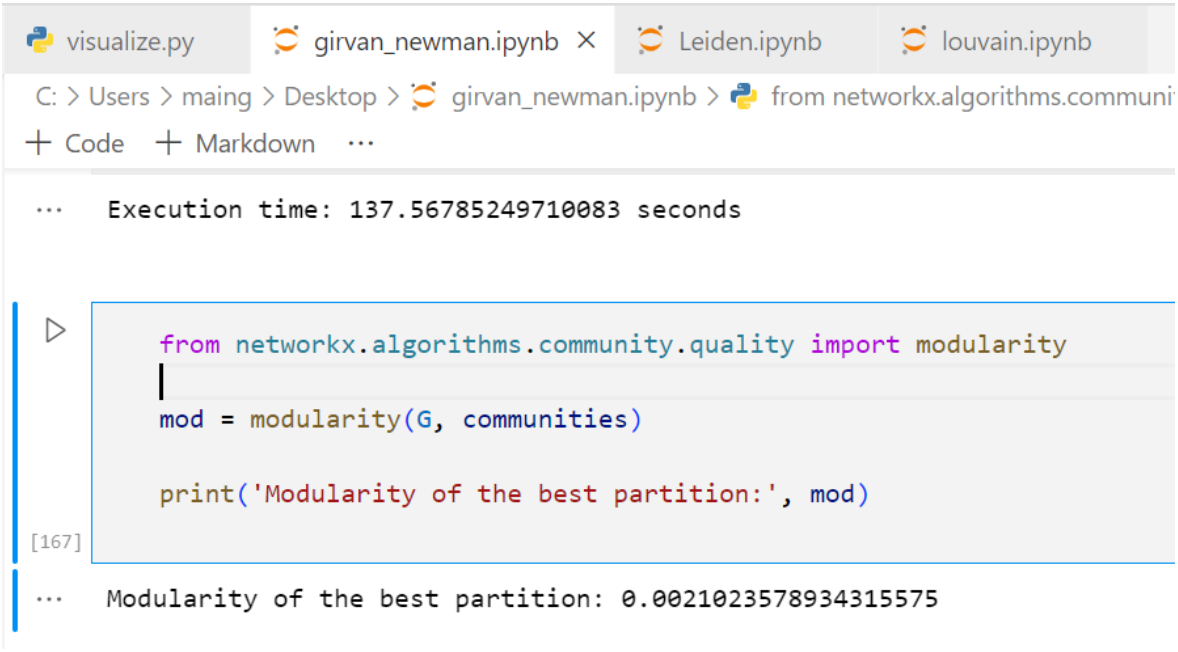

Figure 15: modularity measurement for Girvan Newman algorithms

Louvain: The Louvain algorithm is recognized for its ability to generate community partitions with high modularity scores. Its iterative optimization process helps identify dense subgraphs and optimize modularity locally and globally.

Leiden: The Leiden algorithm slightly higher score than Louvain.

Girvan-Newman: The Girvan-Newman algorithm tends to have a lower modularity score compared to both Louvain and Leiden. This is because its divisive edge removal approach may not capture the community structure as effectively as the other two algorithms in most cases.

4. Time Efficiency comparison

To improve the efficiency of measuring algorithm modularity, we implemented a strategy of dividing the dataset into smaller subsets. Our approach involved selecting the top 500 books that appeared in all book pairs from the previous Map Reduce step. To ensured that all three algorithms were executed under similar conditions with large input size.

```
# print number of edges and nodes
print("Number of edges:", G.number_of_edges())
print("Number of nodes:", G.number_of_nodes())
```

```
Number of edges: 15652
Number of nodes: 500
```

Figure 16: Number of edge and node for time efficiency measurement

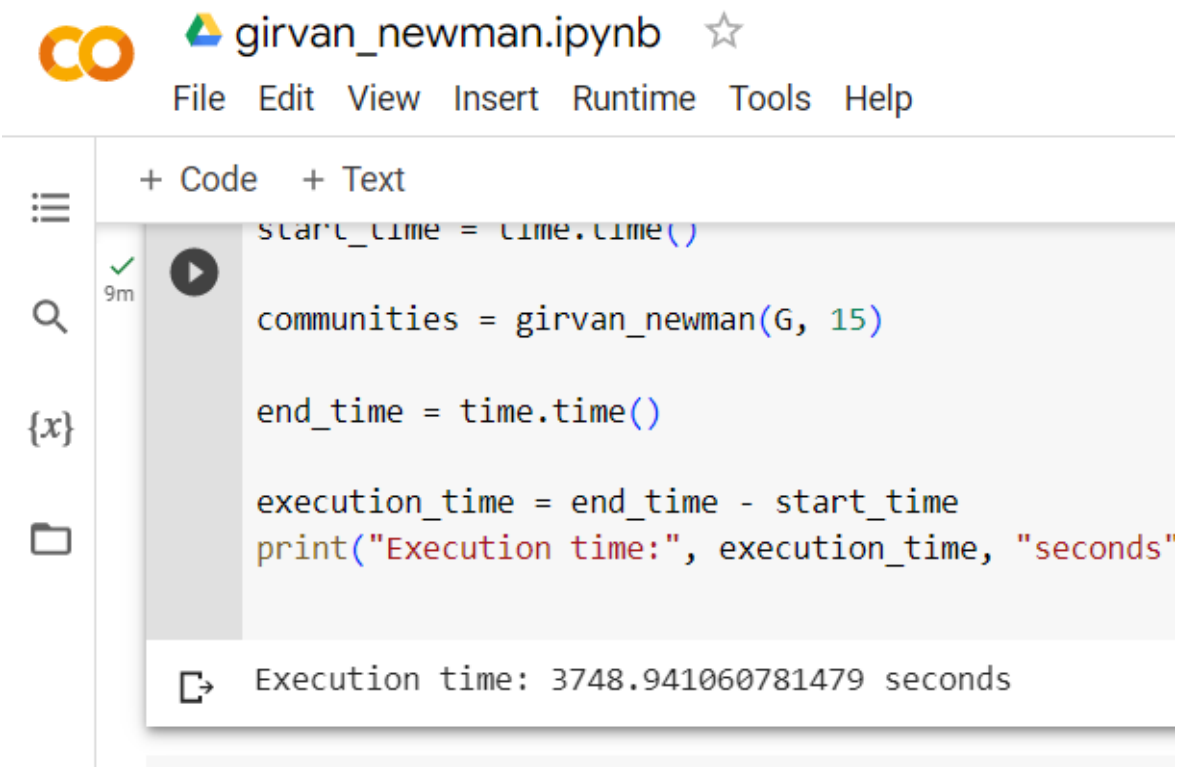

Figure 17: modularity measurement for Girvan Newman algorithms

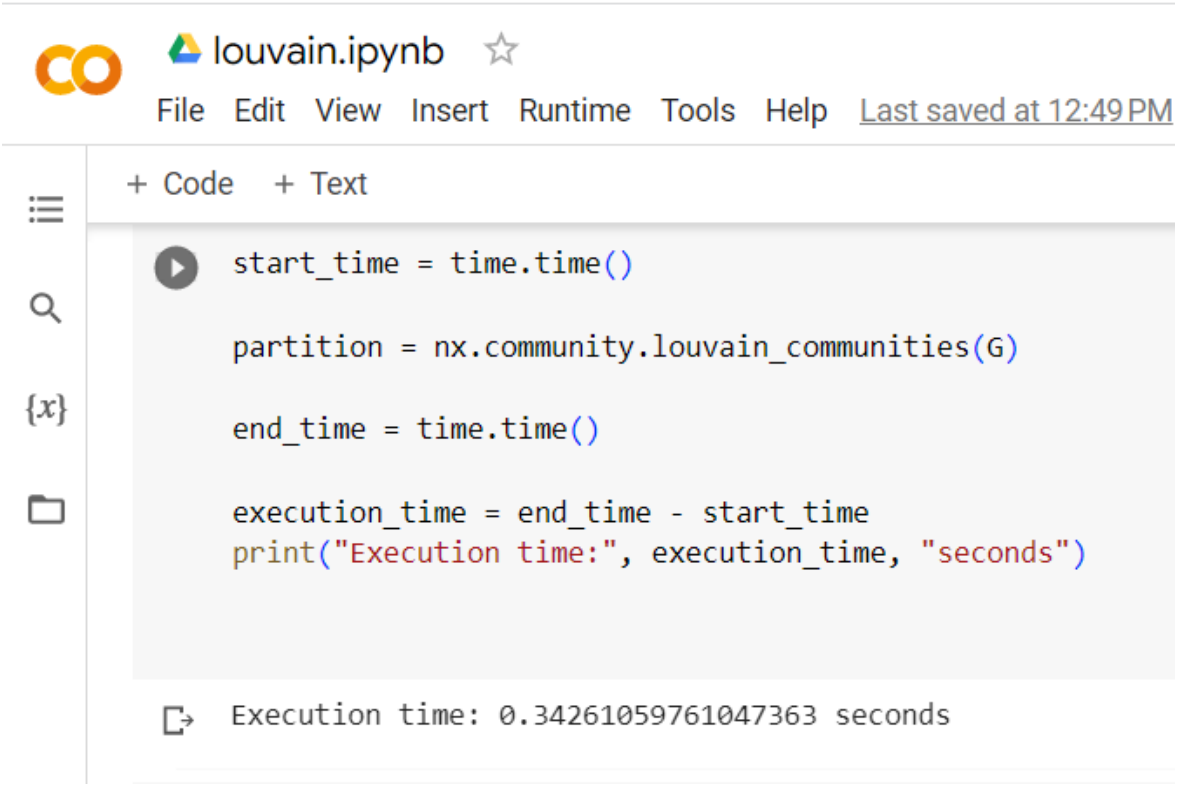

Figure 18: modularity measurement for Louvain algorithms

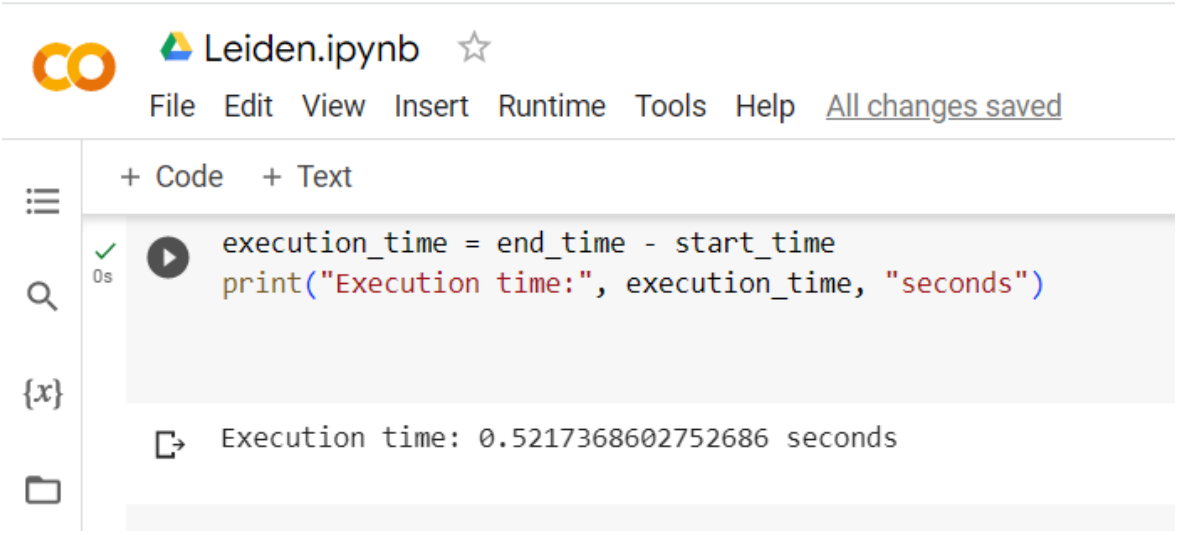

Figure 19: modularity measurement for Leiden algorithms

Leiden: The Leiden algorithm is known for its efficiency, particularly in handling large-scale networks. It utilizes techniques such as smart local moving and graph contraction to improve performance.

Louvain: The Louvain algorithm is also efficient and widely used for community detection. It employs a two-phase approach, iteratively optimizing modularity at the local and global levels. However, it can face scalability challenges

with very large networks. In this comparison, the execution times of Louvain and Leiden are similar, possibly because the Louvain algorithm has been improved with support from libraries or optimized implementations.

Girvan-Newman: The Girvan-Newman algorithm uses a divisive approach, iteratively removing edges with the highest betweenness centrality to break the graph into communities. This approach is significantly more computationally expensive compared to Louvain and Leiden (3748 seconds), especially for large networks. This is because calculating edge betweenness centrality for all edges can be time-consuming and resource-intensive.

5. Overall comparison

We have the result of each algorithms:

	Girvan Newman	**Louvain**	**Leiden**
Modularity	0.0021	0.2714	0.2750
Time(s)	3748.9411	0.3426	0.5217
Number of communities	12	21	11

In conclusion, the Louvain and Leiden algorithms generally perform well in terms of modularity score and time efficiency. They are efficient and effective for community detection in various network sizes. However, the Girvan-Newman algorithm, while effective, may not achieve as high modularity scores and much more computationally expensive. It is important to consider the specific characteristics of the network and the desired trade-offs between modularity and runtime when selecting an appropriate algorithm for community detection.

V. Conclusion

1. Achievement

The algorithms that the group used to solve the community search problem for the artist dataset above accomplished the following:

- Our team have implemented and runned 3 algorithms (Girvan Newman, Louvain, Leiden) on Python, from that, we can see some comparisions and how they works.
- Using the above data clusters, we can suggest similar books to the user while they are seeing the information of the books they want to buy, increasing the diversity of the e-commerce website such as Tiki.
- Analyzing the tastes of the readers community in specific clusters to promote better suggestions and apply them to run ads or PR is another strength that this algorithm brings.
- Besides, we also build a website to demonstrate our work and its application. The demonstration video of the website is here: `https://youtu.be/nnhuGfggjb4`

2. Drawback

Because of the limited research time, the group did not generate many algorithms or make very specific comparisons with the algorithms sought.
However, with the above three algorithms,Girvan-Newmana and Louvain, classical algorithms, and Leiden, an improved algorithm widely used today, the group has also made appropriate comparisons of performance and reliability to generalize them.

3. Future work

The team plans to use more algorithms in the future to get a thorough understanding of the approaches used by businesses and researchers to address problems.

The team also intends to work with more actual data sets in order to fully assess the algorithm' s efficacy and its potential for use in other contexts. This deals with, for instance, developing techniques for providing suggestions or advertising using these data clusters.

References.

[1] Dataset source: https://www.kaggle.com/datasets/arashnic/book-recommendation-dataset

[2] Lilian Weng, Filippo Menczer Yong-Yeol Ahn (2013). "Virality Prediction and Community Structure in Social Networks". Last access 26/05/2023 https://www.researchgate.net/publication/256188948_Virality_Prediction_and_Community_Structure_in_Social_Networks

[3] Fabian Nguyen (2021). "Leiden-Based Parallel Community Detection". Last access 26/05/2023 https://i11www.iti.kit.edu/_media/teaching/theses/ba-nguyen-21.pdf

[4] How Authors Can Find Their Ideal Reading Audience. Last access 26/05/2023 https://www.janefriedman.com/how-authors-can-find-readers/