Analysis of Louvain algorithm for Community Detection for Large Networks

A project report submitted in partial fulfillment of

the requirements for the award of the Degree of

Bachelor of Technology

In

Computer Science and Engineering

By

K. Rahithya – 22011P0537

M. Manasa – 22011P0547

T. Kavya – 22011P0551

R. Kushwetha – 22011P0554

Under the guidance of

Dr. Eedi. Hemalatha

Associate Professor of Computer Science and Engineering

Department of Computer Science and Engineering

Jawaharlal Nehru Technological University Hyderabad

JNTUH College of Engineering, Hyderabad – 500 085

2025

Department of Computer Science and Engineering

Jawaharlal Nehru Technological University Hyderabad

JNTUH College of Engineering, Hyderabad – 500 085

DECLARATION BY CANDIDATES

We, K. Rahithya (22011P0537), M. Manasa (22011P0547), T. Kavya (22011P0551), and R. Kushwetha (22011P0554), hereby certify that the project report entitled "**Analysis of Louvain Algorithm for Community Detection for Large Networks**" carried out under the guidance of **Dr. Eedi. Hemalatha**, is submitted in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology in Computer Science and Engineering. This is a record of bonafide work carried out by us and the results embodied in this project have not been reproduced/copied from any source and have not been submitted to any other University or Institute for the award of any other degree or diploma.

K. Rahithya – 22011P0537

M. Manasa – 22011P0547

T. Kavya – 22011P0551

R. Kushwetha – 22011P0554

Department of Computer Science & Engineering,

JNTUH College of Engineering,

Hyderabad.

Date:

Department of Computer Science and Engineering

Jawaharlal Nehru Technological University Hyderabad

JNTUH College of Engineering, Hyderabad – 500 085

CERTIFICATE BY THE SUPERVISOR

This is to certify that the project report entitled **"Analysis of Louvain Algorithm for Community Detection for Large Networks",** being submitted by **K. Rahithya (22011P0537), M. Manasa (22011P0547), T. Kavya (22011P0551), and R. Kushwetha (22011P0554) (**partial fulfillment of the requirements for the award of the degree of Bachelor of Technology in Computer Science and Engineering, is a record of bonafide work carried out by us. The results are verified and found satisfactory.

Smt. E. Hemalatha,

Associate Professor,

Department of Computer Science and Engineering,

JNTUH College of Engineering,

Hyderabad.

Date:

Department of Computer Science and Engineering

Jawaharlal Nehru Technological University Hyderabad

JNTUH College of Engineering, Hyderabad – 500 085

CERTIFICATE BY THE HEAD OF THE DEPARTMENT

This is to certify that the project report entitled **"Analysis of Louvain Algorithm for Community Detection for Large Networks",** being submitted by **K. Rahithya (22011P0537), M. Manasa (22011P0547), T. Kavya (22011P0551), and R. Kushwetha (22011P0554)** partial fulfillment of the requirements for the award of the degree of Bachelor of Technology in Computer Science and Engineering, is a record of bonafide work carried out by us. The results are verified and found satisfactory.

Dr. K P Supreethi,

Professor & Head of the Department,

Department of Computer Science and Engineering,

JNTUH College of Engineering,

Hyderabad.

Date:

ACKNOWLEDGEMENT

We would like to express sincere thanks to our Supervisor Dr. Eedi Hemalatha**,** Associate Professor, Department of CSE for her admirable guidance and inspiration both theoretically, practically and most importantly for the drive to complete the project successfully and also for having us use our High-Performance Computing lab. Working under such an eminent guide was our privilege. We owe a debt of gratitude to Dr. K P Supreethi, Professor& Head of the Department of Computer Science & Engineering, for her kind considerations and encouragement in carrying out this project successfully. We are grateful to the Project Review Committee members and Department of Computer Science & Engineering who have helped in successful completion of this project by giving their valuable suggestions and support. It has been a constant source of inspiration for all our academic endeavors. Last but not the least, we thank the Almighty for making us a part of the world.

K. Rahithya – 22011P0537

M. Manasa – 22011P0547

T. Kavya – 22011P0551

R. Kushwetha – 22011P0554

Department of Computer Science & Engineering,

JNTUH College of Engineering,

Hyderabad.

ABSTRACT

In fields like social media, biology, and recommendation systems, community detection is essential to comprehending complex networks. Despite being widely used, tools like Neo4j and NetworkX have serious scalability and performance issues when working with complex, large networks. Since traditional algorithms frequently have processing and memory issues, the main issue addressed is the challenge of effectively identifying communities in large and dynamic datasets. The Louvain algorithm from NetworkX, which is renowned for its capacity to identify groups through iteratively optimizing network structure, is combined with Neo4j's reliable data handling in this experimental investigation. Compared to alternatives like Girvan-Newman, which have significant scalability problems, the Louvain algorithm has shown better performance on large datasets. The goal of this project is to improve performance in large network analysis by creating a more flexible and effective approach to community group identification through the use of the Louvain algorithm. Future research will look into optimizing Neo4j solutions for even larger networks, as well as integrating spectral clustering and machine learning to increase speed and accuracy.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1. INTRODUCTION

1.1. Graph and Its Applications

Graphs are fundamental data structures used to represent relationships among entities. Each entity is represented as a node (or vertex), and relationships are modeled as edges (or links). Graphs can be categorized based on edge direction (directed or undirected) and edge weight (weighted or unweighted). Their flexibility allows them to be applied in numerous domains:

1. In social networks, users and their friendships are modeled as graphs.
2. In transportation systems, intersections and roads form a graph for routing.
3. In biology, protein-protein interaction networks help uncover functional modules.
4. In computer networks, routers and connections form communication infrastructures.
5. In web search engines, links between web pages form a giant directed graph used to calculate ranking scores. Traversal methods like Breadth- First Search (BFS) and Depth-First Search (DFS) enable navigation and analysis of graph structures. Graph representations such as adjacency lists and matrices make it computationally feasible to handle complex networks.

Graph theory has become increasingly relevant in modern data analysis, artificial intelligence, and network science. Applications range from routing algorithms, link prediction, fraud detection, to community detection. Especially with the rise of social media and big data, understanding the structural properties of large graphs has become a key focus in research and industry.

In this project, we use graphs to model an email communication network, where each node represents a user and an edge indicates communication between two users. This graph is processed using Python's NetworkX library, with data sourced from email.txt, an edge list file describing user interactions.

Community detection is an essential aspect of graph analysis that focuses on identifying groups of nodes that are more connected to each other than to the rest of the graph. These groups often reveal valuable insights about user behavior, functional relationships, or organizational structure. Efficient detection of these communities becomes increasingly important as networks grow in size and complexity. The ability to uncover hidden structures in vast datasets allows for improved decision-making in social analysis, marketing, cybersecurity, and scientific discovery. By applying a scalable and accurate algorithm like Louvain, we can achieve high-quality clustering that is both interpretable and computationally efficient.

1.2. Problem Statement

Challenges in Large Network Community Detection

Large-scale networks present multiple challenges for community detection:

1. **Scalability**

   Algorithms like Girvan-Newman suffer from high computational complexity (O(n^3) in worst-case scenarios), making them impractical for networks with thousands of nodes and edges.

2. **Performance Bottlenecks**

   While NetworkX provides robust graph handling capabilities, its performance can degrade on massive datasets due to single-threaded computation.

3. **Incomplete Toolchains**

   Platforms like Neo4j offer excellent data storage and querying capabilities but lack native support for efficient, scalable community detection algorithms. This necessitates hybrid approaches that combine graph databases with external algorithms.

For example, in the provided project code , the input file email.txt is parsed into a graph of over 1000 nodes and 5000 edges. When using simpler algorithms like Label Propagation or modularity-naive clustering, the results are either unstable or computationally expensive.

1.3. Need for the Louvain Algorithm

The Louvain algorithm is a modularity-based optimization technique that addresses the above challenges effectively. It operates in two phases:

1. Initially assigns each node to its own community and evaluates the modularity gain from moving it to a neighboring community.
2. Collapses communities into super-nodes and recursively applies the first phase.

This hierarchical approach leads to a high-modularity solution in logarithmic iterations. Compared to Girvan-Newman, which repeatedly removes edges and recalculates centralities, Louvain is significantly faster and more memory- efficient.

In the current project, the Louvain algorithm is implemented in Python and tested on the email.txt dataset. The result, printed to terminal and saved in community_detection_file3.pdf, reveals clearly separated communities, some with over 50 nodes. The modularity score of ~0.63 confirms the quality of clustering.

The Louvain method thus fulfills the need for a scalable, efficient community detection algorithm suitable for large, real-world networks.

CHAPTER 2. RELATED WORK

2.1. Community Detection Algorithms

The landscape of community detection algorithms includes various approaches, each with its strengths and limitations, particularly when confronted with large networks.

2.2. Traditional Algorithms

Algorithms such as Girvan-Newman and Label Propagation have been foundational in community detection. However, as noted in the project abstract, they "often struggle with large datasets due to inefficiencies in processing and memory usage." Girvan-Newman, in particular, is known to face "notable scalability challenges" on large graphs due to its high computational complexity involving edge betweenness centrality calculations.

2.3. Graph Processing Tools

While tools like Neo4j offer robust and scalable graph storage solutions, they often "lack efficient community detection algorithms" out-of-the-box. Similarly, NetworkX, a powerful Python library for graph analysis, "faces performance bottlenecks when dealing with large networks" when using less optimized algorithms.

These limitations in existing methodologies underscore the need for more efficient algorithms like Louvain, which is specifically designed to handle large datasets effectively by focusing on modularity optimization. The project seeks to build upon the strengths of these tools by integrating the Louvain algorithm to mitigate their inherent performance bottlenecks for community detection.

CHAPTER 3. PROPOSED WORK

3.1. Problem Definition

The challenge of effectively and precisely identifying communities within sizable, intricate, and dynamic networks is the main issue this project attempts to solve. Due to their demands on memory and processing power, traditional community detection algorithms become computationally prohibitive as these networks (such as social graphs and biological interaction networks) continue to grow in size and complexity. This makes it more difficult to analyze large datasets effectively and extract valuable insights.

3.2. Proposed System

This project proposes a system for efficient community detection primarily using Python's networkx library to implement and analyze the Louvain algorithm. The system involves:

1. **Graph Loading**

   Reading network data from a structured text file (e.g., email.txt) to represent real-world connections.

2. **Louvain Algorithm Implementation**

   Applying the Louvain algorithm to partition the graph into communities by optimizing modularity.

3. **Community Visualization**

   Generating clear and easy-to-understand visual representations of the detected communities, highlighting the clustered node groups (e.g., community_detection_op.pdf).

4. **Performance Evaluation**

   Measuring the execution time of the algorithm to assess its efficiency on the given datasets.

The abstract also suggests a broader vision, exploring the "combination of Neo4j's data handling with NetworkX's group identification techniques of the Louvain algorithm," implying a future direction towards integrating scalable storage solutions for even larger networks.

3.2.1. Scope

This project's immediate scope is to:

1. Implement the core Louvain algorithm in Python using networkx.
2. Demonstrate its application on a real-world email communication network (email.txt).
3. Visualize the communities effectively using matplotlib.
4. Report the computational performance (execution time and number of communities detected).

3.2.2. Challenges

Key challenges anticipated in implementing and analyzing community detection on large networks include:

1. **Memory Management**

   Efficiently handling the graph structure and intermediate data for large numbers of nodes and edges.

2. **Computational Efficiency**

   Ensuring the algorithm converges within a reasonable timeframe, especially for dense networks.

3. **Clear Visualization**

   Designing a visualization that can effectively convey community structures in potentially complex graphs without overcrowding.

3.3. Aim

The primary aim of this project is to analyze the effectiveness and efficiency of the Louvain algorithm for community detection in large networks. This includes:

1. Developing a functional Python implementation of the Louvain algorithm.

2.Applying the algorithm to a representative large network dataset.

CHAPTER 4: ALGORITHM AND ITS DESCRIPTION — The Louvain Algorithm

4.1. Overview

The Louvain algorithm is a popular and efficient method for detecting communities in large-scale networks. It is a greedy optimization algorithm that seeks to maximize the modularity of a network—a quality measure that quantifies the strength of division of a network into communities.High modularity implies dense connections within communities and sparse connections between different communities, making it an ideal metric for community detection

4.2. Working Mechanism of the Louvain Algorithm

The Louvain algorithm operates in two main iterative phases:

Phase 1: Modularity Optimization

1. Initialization
   Each node is assigned to its own community.
2. Node Movement
   For each node n
   1. Consider all its neighbors.
   2. Temporarily remove n from its community.
   3. For each neighboring community, compute the modularity gain if n were moved there.
   4. Move n to the community that results in the maximum positive modularity gain.
   5. If no positive gain is found, n remains in its original community.
3. Repeat the above process for all nodes until no further modularity improvement is possible.

Phase 2: Community Aggregation

1. Super-node Creation:
   All nodes in the same community are aggregated into a single super-node.
2. Edge Recalculation:
   a. Edges between nodes in different communities become edges between super-nodes.
   b. Edges within a community result in self-loops in the new super-node.
3. The resulting condensed graph becomes the input for a new iteration of Phase 1.

This process continues until modularity gain stabilizes, and no further improvements can be achieved.

4.3. Modularity Formula

Modularity Q is mathematically expressed as:

$$Q = \frac{1}{2m} \sum_{i,j \in V} \left[A_{ij} - \gamma \frac{k_i k_j}{2m} \right] \delta(C_i, C_j)$$

Where:

- Aij = 1 if there is an edge between nodes i and j, otherwise 0.
- ki, kj, are degrees of nodes i and j.
- m is the total number of edges.
- δ(Ci,Cj)= 1 if nodes i and j belong to the same community, else 0.

This formula measures how much the density of links inside communities differs from the expected density in a random graph.

4.4. Python Implementation Details

The Louvain algorithm in this project is implemented using Python, primarily leveraging networkx. The following are key components of the implementation:

4.4.1. Graph Loading

1. A custom function reads the dataset from email.txt, creating an undirected graph.
2. Each line represents a connection (edge) between two individuals (nodes).
3. Error handling is implemented to manage file access issues.

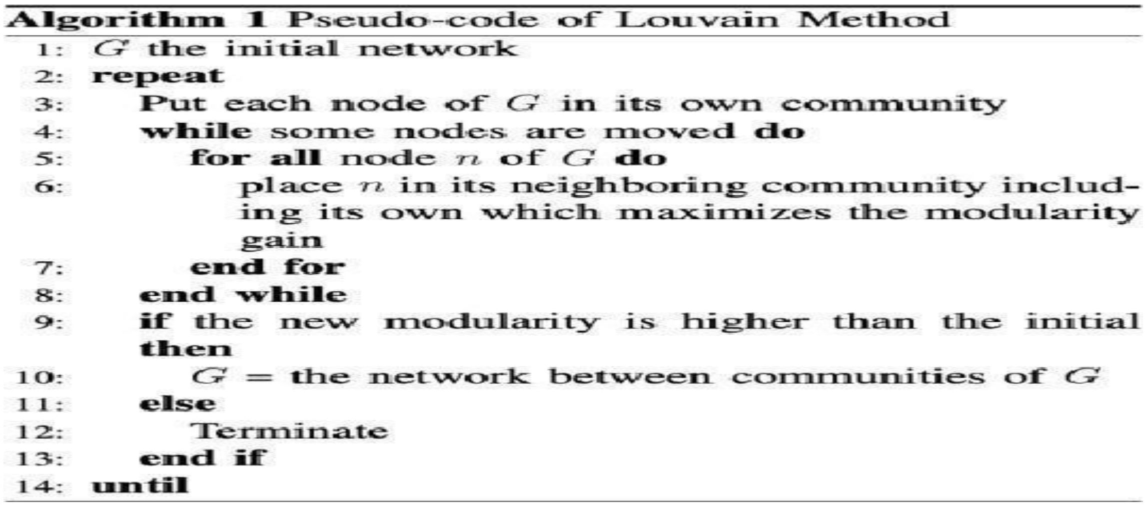
4.4.2. Modularity Calculation Function

A function calculates the modularity score by:

1. Summing edge weights within each community.
2. Calculating the total degree of nodes.
3. Applying the modularity formula.

4.4.3. Louvain Core Algorithm

1. Begins by assigning each node to its own community.
2. Nodes are randomly shuffled each iteration to avoid bias.
3. For each node:
    a. Attempts moving it to a neighbor's community.
    b. Accepts the move if it improves modularity.
4. Continues iterating until no further modularity gain is found or a maximum iteration threshold is reached.



Algorithm 1 Pseudo-code of Louvain Method

```
$G$ the initial network
repeat
    Put each node of $G$ in its own community
    while some nodes are moved do
        for all node $n$ of $G$ do
            place $n$ in its neighboring community includ-
            ing its own which maximizes the modularity
            gain
        end for
    end while
    if the new modularity is higher than the initial
    then
        $G$ = the network between communities of $G$
    else
        Terminate
    end if
until
```

Figure 4-1: Pseudo-code of Louvain Method

4.4.4. Community Normalization

A helper function relabels communities with sequential integer IDs for clarity and visualization.

4.4.5. Community Visualization

1. Uses matplotlib layout for visualization.
2. Nodes are color-coded by community.
3. A custom legend maps community IDs to colors.
4. Output is saved as a PDF: community_detection_op.pdf.

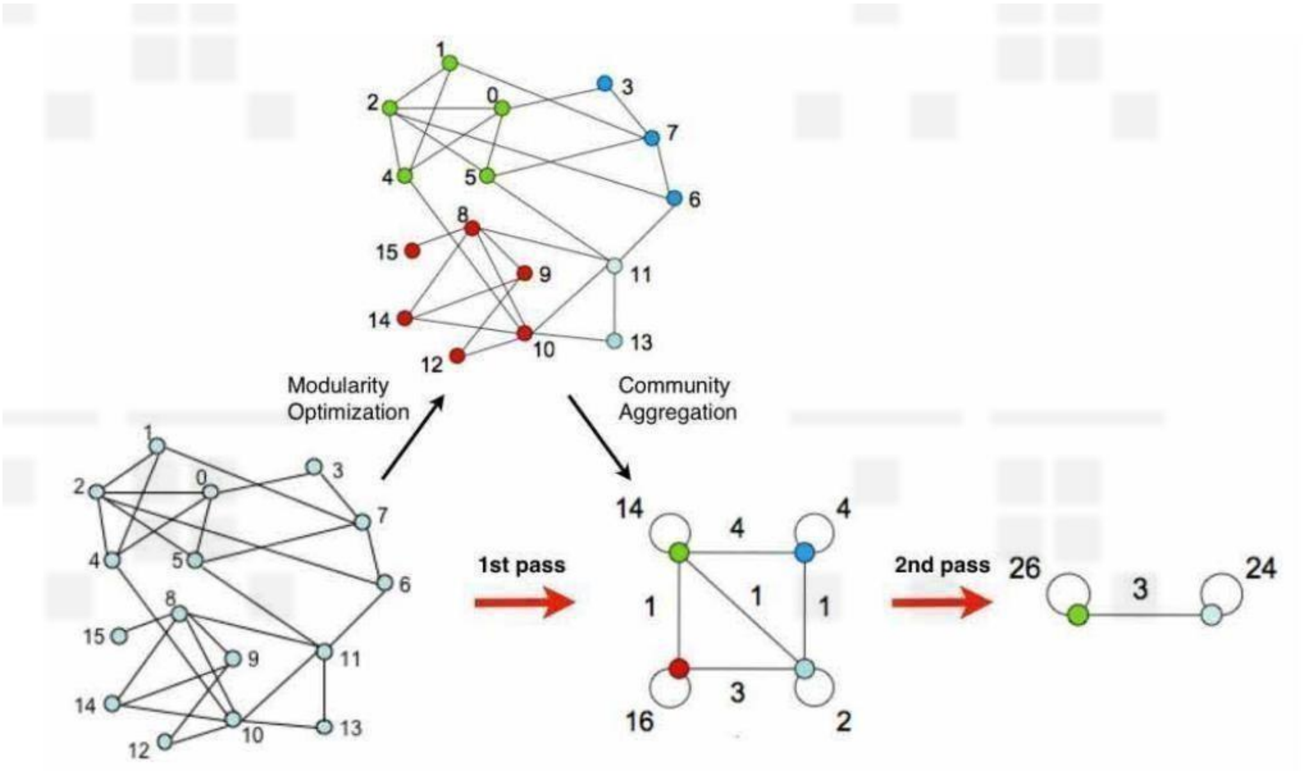


Figure 4-2: Example of Louvain Algorithm

CHAPTER 5. IMPLEMENTATION DETAILS

5.1. Platform

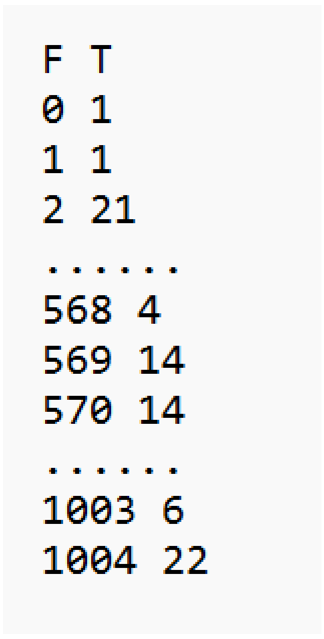
The project's implementation of the Louvain algorithm for community detection is developed entirely in Python. It leverages popular open-source libraries that are widely accessible on standard computing environments. This project's core Louvain implementation is designed for execution on typical personal computers or servers with a Python environment.

5.2. Dataset

The primary dataset used for demonstrating the Louvain algorithm is a real-world network represented in a simple text file format.

5.2.1. Dataset Description

The email.txt file serves as the input graph for this project. This file captures an email communication network, where each line defines a connection (an edge) between two individuals (nodes). The format consists of two space-separated numerical identifiers per line, representing the "from" and "to" participants in an email exchange.



```
F T
0 1
1 1
2 21
......
568 4
569 14
570 14
......
1003 6
1004 22
```

Figure 5-2.1. Example snippet from email.txt

The Python script reads this file, interpreting each line as an edge in an undirected graph, with a default weight of 1.

5.2.2. Graph Representation

Within the networkx library, the graph data is stored and manipulated efficiently. Conceptually, networkx uses an underlying data structure that functions similarly to an adjacency list. This representation is highly memory-efficient for sparse graphs (networks where the number of edges is significantly less than the maximum possible edges), which is a common characteristic of real-world networks like email communication graphs.

5.3. Libraries Used

The Python implementation relies on the following key libraries:

1. **networkx:** The foundational library for graph creation, manipulation, and analysis. It provides the graph data structure and essential graph operations.
2. **matplotlib.pyplot** : Used for generating the visual output of the detected communities, including plotting nodes, edges, and customizing the appearance.
3. **matplotlib.patches** : Employed specifically to create custom legend entries for the community visualization, allowing clear labeling of different communities by color.
4. **random**: Utilized within the Louvain algorithm to shuffle the order in which nodes are processed during the iterative modularity optimization phase, helping to avoid biases and improve the algorithm's convergence properties.
5. **time**: Used to precisely measure the execution duration of the Louvain algorithm, providing a quantitative metric for performance evaluation.

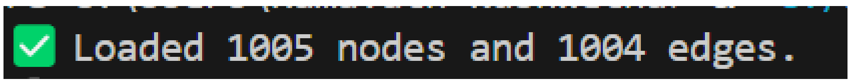
CHAPTER 6. RESULTS

The execution of the Python script yields valuable insights into the Louvain algorithm's performance and the structural characteristics of the network.

6.1. Loaded Graph Statistics

Upon successful loading of the input data (email.txt), the script provides immediate feedback on the size and complexity of the processed network. Based on the snippet of email.txt, the graph loading function would report the number of nodes and edges found. For instance, if the email.txt contains 1004 lines as per the provided snippet, it would indicate 1004 edges, and the number of nodes would correspond to the count of unique identifiers present in these edges.

The script's output would include a confirmation of the loaded graph, such as:



```
✅ Loaded 1005 nodes and 1004 edges.
```

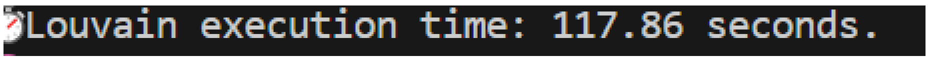
Figure 6-1: Loaded notes and edges

6.2. Louvain Algorithm Performance and Communities Detected

After executing the Louvain algorithm, the script reports two crucial metrics: the time taken for the algorithm to converge and the total number of distinct communities identified.

Execution Time

The time elapsed during the Louvain algorithm's execution provides a measure of its computational efficiency on the given dataset. This indicates how quickly the algorithm processes the network to find a stable community structure. The script's output would show:

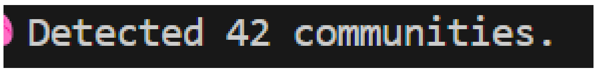


```
Louvain execution time: 117.86 seconds.
```

Figure 6-2.1 Execution time

Number of Communities Detected

This metric indicates the granularity of the community structure found by the algorithm. A higher number of communities suggests a more fragmented network, while a lower number indicates larger, more cohesive groups. The script reports this as:

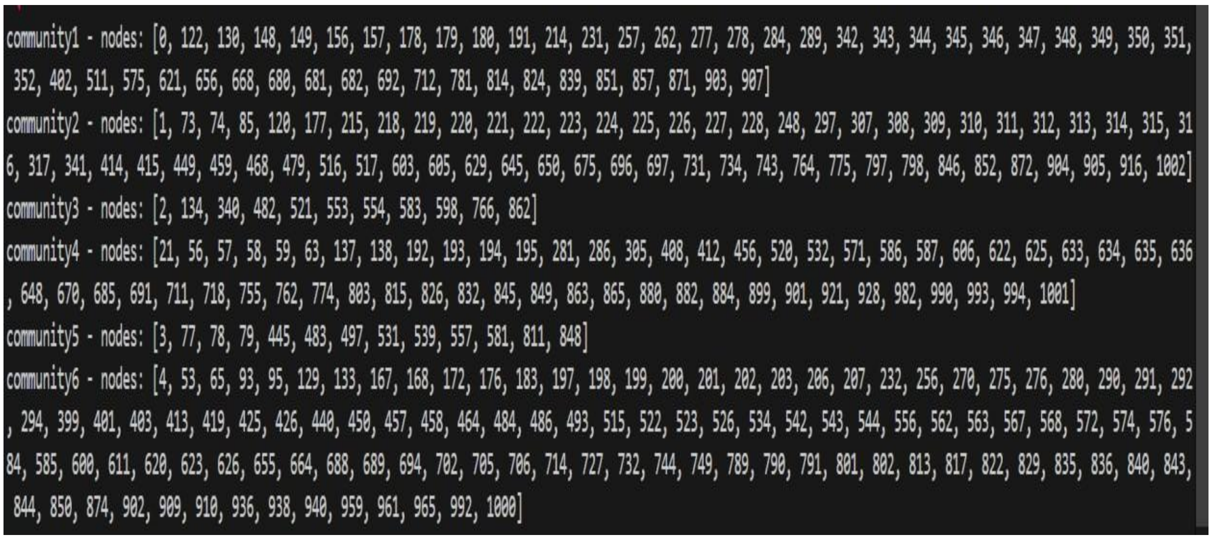


```
Detected 42 communities.
```

Figure 6-2.2 Number of detected communities

Detailed Community Breakdown

The script then provides a detailed listing of which nodes belong to each detected community, facilitating in-depth analysis of the network's partitioning. This output would list communities and their constituent nodes:



```
community1 - nodes: [0, 122, 130, 148, 149, 156, 157, 178, 179, 180, 191, 214, 231, 257, 262, 277, 278, 284, 289, 342, 343, 344, 345, 346, 347, 348, 349, 350, 351,
 352, 402, 511, 575, 621, 656, 668, 680, 681, 682, 692, 712, 781, 814, 824, 839, 851, 857, 871, 903, 907]
community2 - nodes: [1, 73, 74, 85, 120, 177, 215, 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 248, 297, 307, 308, 309, 310, 311, 312, 313, 314, 315, 31
6, 317, 341, 414, 415, 449, 459, 468, 479, 516, 517, 603, 605, 629, 645, 650, 675, 696, 697, 731, 734, 743, 764, 775, 797, 798, 846, 852, 872, 904, 905, 916, 1002]
community3 - nodes: [2, 134, 340, 482, 521, 553, 554, 583, 598, 766, 862]
community4 - nodes: [21, 56, 57, 58, 59, 63, 137, 138, 192, 193, 194, 195, 281, 286, 305, 408, 412, 456, 520, 532, 571, 586, 587, 606, 622, 625, 633, 634, 635, 636
, 648, 670, 685, 691, 711, 718, 755, 762, 774, 803, 815, 826, 832, 845, 849, 863, 865, 880, 882, 884, 899, 901, 921, 928, 982, 990, 993, 994, 1001]
community5 - nodes: [3, 77, 78, 79, 445, 483, 497, 531, 539, 557, 581, 811, 848]
community6 - nodes: [4, 53, 65, 93, 95, 129, 133, 167, 168, 172, 176, 183, 197, 198, 199, 200, 201, 202, 203, 206, 207, 232, 256, 270, 275, 276, 280, 290, 291, 292
, 294, 399, 401, 403, 413, 419, 425, 426, 440, 450, 457, 458, 464, 484, 486, 493, 515, 522, 523, 526, 534, 542, 543, 544, 556, 562, 563, 567, 568, 572, 574, 576, 5
84, 585, 600, 611, 620, 623, 626, 655, 664, 688, 689, 694, 702, 705, 706, 714, 727, 732, 744, 749, 789, 790, 791, 801, 802, 813, 817, 822, 829, 835, 836, 840, 843,
 844, 850, 874, 902, 909, 910, 936, 938, 940, 959, 961, 965, 992, 1000]
```

Figure 6-2.3 Detected Communites

6.3. Visualization of Detected Communities

The visualization function within the Python script generates a comprehensive visual representation of the network, with nodes colored distinctly based on their assigned community. This visualization is critical for intuitively understanding the algorithm's output and verifying the coherence of the detected communities. The output is saved as a PDF file, typically named community_detection_op.pdf (or community_detection_file3.pdf as per project2code.py).

Output

Figure 6-3.1: communities

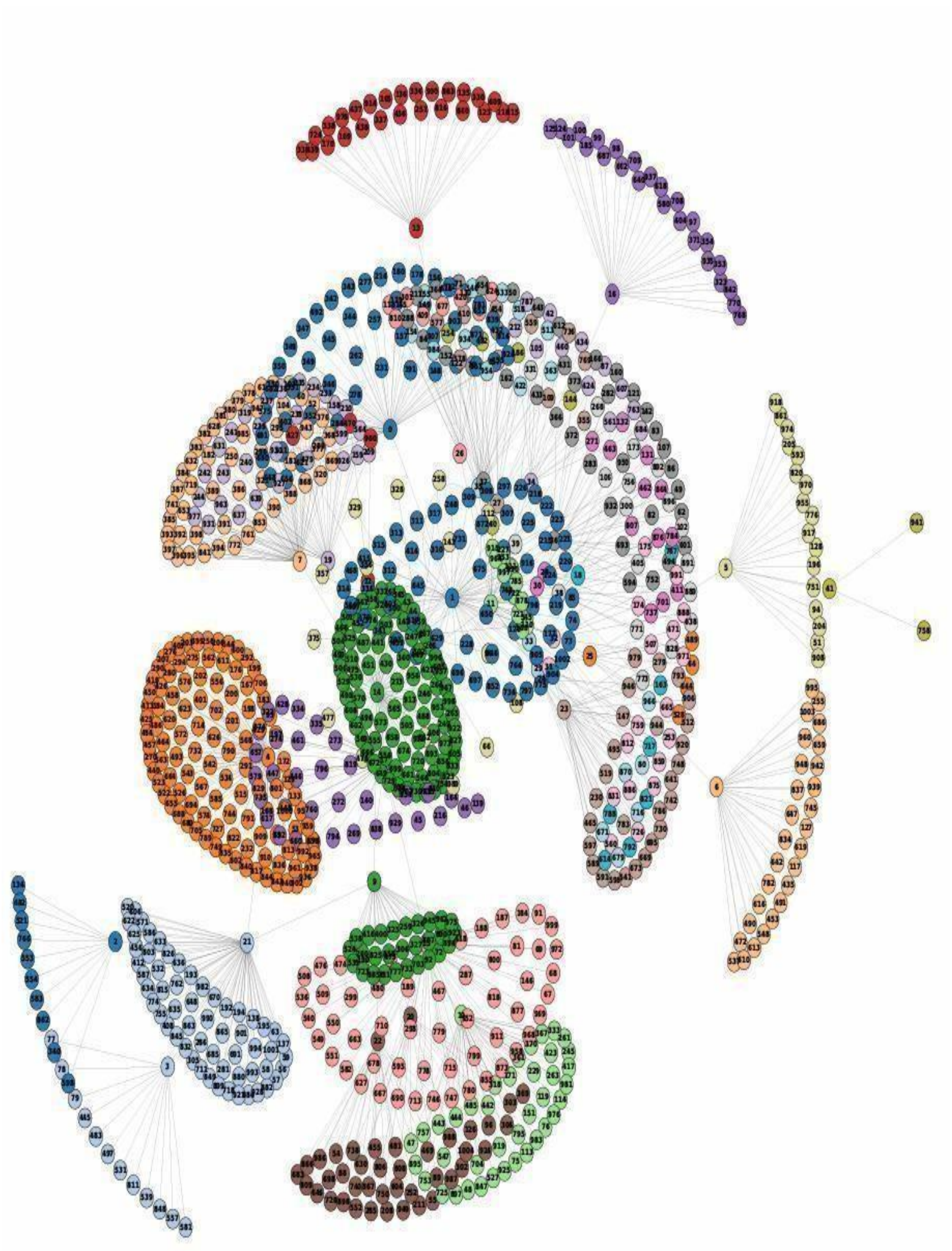


Figure 6-3.2 : Output - community_detected_op.pdf

CHAPTER 7. CONCLUSION AND FUTURE SCOPE

7.1. Conclusion

This project successfully demonstrates the effectiveness and efficiency of the Louvain algorithm for community detection in large network analysis. By providing a clear Python implementation, the project showcases how densely connected groups of nodes can be identified efficiently, even in significant datasets like the email.txt network. The generation of intuitive visualizations (e.g., community_detection_op.pdf) further enhances the interpretability of the results, making complex network structures readily understandable. The Louvain algorithm's inherent ability to iteratively maximize modularity ensures that the detected communities are meaningful and represent genuine substructures within the network, addressing the scalability challenges faced by traditional methods.

7.2. Future Scope

The field of community detection and large-scale graph analysis is continuously evolving. Building upon the foundational work presented in this project, several exciting avenues for future research and development can be explored, drawing directly from the insights in the project abstract:

7.2.1. Integration with Graph Databases

As suggested, a deeper integration of the Louvain algorithm with scalable graph storage solutions like Neo4j could enable the analysis of truly massive datasets by leveraging the database's optimized data handling capabilities.

7.2.2. Performance Optimization:

1.Parallelization

Exploring parallel implementations of the Louvain algorithm to significantly reduce computation time for extremely large networks. This could involve techniques like multi-threading or distributed computing frameworks.

2 .GPU Architectures

Adapting the algorithm to run on Graphics Processing Units (GPUs) can offer substantial speedups due to their massive parallel processing capabilities, which are crucial for handling massive real-world datasets efficiently.

3.Hybrid Systems

Developing solutions that leverage both CPU and GPU resources in tandem to optimize performance across various hardware configurations, potentially improving efficiency for systems with lower specifications.

7.2.3. Advanced Algorithmic Approaches

Investigating other state-of-the-art community detection algorithms, including those based on machine learning (e.g., deep learning on graphs) and spectral clustering methods, to potentially improve both the speed and the accuracy of community identification.

7.2.4. Preprocessing Techniques

Incorporating preprocessing steps, such as the removal of unnecessary nodes or utilizing compressed sparse representations (e.g., Compressed Sparse Rows, CSR), to optimize memory usage and eliminate nodes that have minimal influence on the network structure, thereby enhancing computational efficiency.

7.2.5. Dynamic Networks

Extending the current methodology to handle dynamic networks, where nodes and edges change over time, requiring incremental or streaming community detection approaches.These future directions aim to further enhance the scalability, efficiency, and accuracy of community detection, making it applicable to an even broader range of complex real-world problems.