Experiment 8: Social Network Analysis Using R with Community Detection Algorithms

Date of Performance :

Date of Submission :

1. Statement of problem:

1.1 AIM:

To explore and analyze social structures in complex networks using R, with a specific focus on detecting and interpreting communities using computational algorithms to support domain-specific decision-making and insights.

1.2 OBJECTIVES:

1. To acquire and preprocess social network data from real-world or simulated sources.
2. To model the data into graph structures suitable for computational analysis in R.
3. To visualize network topology and identify structural patterns using graph plotting tools.
4. To apply and analyze the results of different community detection algorithms (e.g., Louvain, Walktrap, Label Propagation, Edge Betweenness).
5. To evaluate detected communities based on modularity and other network quality metrics.
6. To interpret community structures in the context of specific application domains (e.g., marketing, education, research, communication, governance).
7. To compare algorithmic performance and community stability across datasets.
8. To derive actionable insights from community patterns relevant to the domain of the network.

1.3 SCOPE OF THE STUDY

This study focuses on the computational analysis of social networks through community detection, bounded within the use of the **R programming environment** and applicable graph libraries such as igraph, ggraph, and tidygraph.

- **Data Scope:** The networks analyzed will be drawn from digital social domains (e.g., Twitter, academic co-authorship, email networks, online discussion forums). Each

dataset is curated, cleaned, and transformed into graph structures, aligning with Objective 1 & 2.

- **Algorithmic Scope:** The study will employ modularity-based, walk-based, and label-based community detection methods that are scalable and well-supported in R. The evaluation of these methods includes both technical performance and community coherence, implicitly addressing Objectives 4, 5, and 7.
- **Application Scope:** Community analysis will be interpreted in domain-specific contexts:
    - In marketing, to discover influence clusters or niche consumer groups.
    - In education, to detect collaboration patterns among learners or institutions.
    - In research analytics, to map interdisciplinary collaboration.
    - In organizational networks, to reveal internal communication flow and silos.

  These scenarios incorporate Objective 6 and 8 by extending the analysis into actionable interpretations within defined domains.
- **Tool and Methodological Scope:** The implementation is restricted to offline/static network analysis (not real-time), using open-source tools in R. The study will not involve deep learning or NLP techniques unless such attributes are already pre-integrated into node metadata. Objective 3 is embedded

This well-defined scope ensures a focused study while embedding the executional steps (objectives) into specific domain operations without redundancy.

2.Theory

2.1 Data Scope:

This research will focus on analyzing various types of digital social networks sourced from platforms such as Twitter, academic publishing databases, email archives, and online discussion forums. These digital social domains are inherently graph-oriented, where users or entities are nodes, and their interactions form edges. Each selected dataset will undergo a rigorous pipeline of curation to ensure relevance and data integrity. The process includes cleansing the data by removing duplicates, correcting inconsistencies, and formatting structural attributes uniformly. Once cleaned, the data will be transformed into graph structures like edge lists or adjacency matrices. This transformation allows for computational modeling using network analysis libraries in R, thereby satisfying the foundational goals outlined in Objectives 1 and 2. These structured datasets can then be used to explore complex network properties such as centrality, modularity, and community detection.

Acquisition and Preprocessing of Social Network Data

Acquiring social network data involves tapping into both open-access data repositories and real-time data APIs offered by platforms. For instance, Twitter provides access to user interaction data such as retweets, mentions, and replies via the Twitter API, which requires API keys and developer access. Reddit and Stack Overflow data can be accessed via the Pushshift API or downloaded directly from public data dumps. Academic co-authorship data is retrievable from databases like DBLP and arXiv, which offer structured bibliographic metadata in XML or JSON formats. For email networks, datasets such as the Enron email corpus are freely available and well-structured for experimentation.

Preprocessing is an essential step that transforms raw data into a form suitable for graph construction. This includes cleaning corrupted or incomplete entries, standardizing date and time formats, mapping IDs, and creating consistent naming conventions. It may also involve removing bot-like activity (in Twitter data), merging author aliases (in co-authorship networks), or anonymizing sensitive information (in email datasets). Once cleaned, the relationships among users are defined to create the edge structure. For instance, a reply on a forum becomes a directed edge from the replier to the original poster. The final stage of preprocessing involves formatting the data into R-readable structures such as CSVs, edge lists, or adjacency matrices.

Various R packages assist in these tasks. For data wrangling, the tidyverse suite provides functions for filtering, transforming, and reshaping data. To interface with web APIs and scrape data, packages like httr, rvest, and jsonlite are employed. These tools help automate data fetching, extraction, and parsing from web sources or APIs.

Modeling Data into Graph Structures for Analysis in R

Once the data is preprocessed, it is essential to model it as a graph. In this context, graph modeling involves representing entities as nodes (or vertices) and their relationships as edges (or links). Depending on the nature of interaction, graphs can be directed, such as in Twitter where user A retweeting user B indicates a directed edge from A to B, or undirected, such as in co-authorship where a mutual collaboration exists. These graph models can be simple (nodes and edges only) or complex with weighted edges and node attributes like user type, post length, or publication date.

Graphs can be represented in several data formats. An edge list records pairs of nodes indicating the existence of a connection. An adjacency matrix provides a tabular view where the presence of a link is marked by a '1' and absence by '0'. Incidence matrices are used for bipartite graphs, such as author-paper networks, where nodes belong to different sets. In more advanced representations, graphs can also include attributes, such as edge weight for frequency of interactions or node attributes like geographic location.

To implement these models in R, the igraph package offers comprehensive functionality for graph construction, manipulation, and visualization. Graphs are typically created using the graph_from_data_frame() function, which takes a data frame of edges and optional vertex attributes. The tidygraph package offers a tidyverse-friendly interface for manipulating graphs, while ggraph is used for aesthetic and layout-rich visualization. Using these tools, one can compute network metrics such as degree centrality (to find influential nodes), betweenness (nodes acting as bridges), clustering coefficient (to understand local density), and modularity (for detecting communities or clusters).

Datasets and Sources

1. Twitter Network

The Twitter network dataset captures user interactions such as mentions, retweets, likes, and replies. These actions create a directed and weighted social graph where influence and connectivity can be analyzed. The Twitter Developer Platform provides access to these interactions via the Twitter API (now X API), requiring OAuth credentials and adherence to usage limits. Researchers may use filtered streams, search endpoints, or historical data via premium endpoints. Alternatively, archived datasets from institutions or the Internet Archive can be used. For example, https://archive.org/details/twitterstream offers public tweet data for academic use.

2. Academic Co-authorship

Academic co-authorship datasets provide insight into research collaboration and the formation of academic communities. Each node represents an author, and an edge represents a co-authored publication. The DBLP dataset offers XML files containing author, title, and publication metadata, which can be parsed using XML parsers in R. Similarly, the arXiv platform provides an API for accessing article metadata including author lists. These networks are typically undirected and can be enriched with attributes like the number of papers co-authored or citation counts. DBLP's XML dataset is accessible at https://dblp.org/xml/, and arXiv's API is available at https://arxiv.org/help/api/.

3. Email Communication

Email networks provide temporal, directed communication patterns between users. The Enron email corpus is a widely used dataset that includes over 500,000 emails from about 150 users, primarily senior management at Enron. Nodes represent email addresses or users, and edges indicate the act of sending or receiving an email. Attributes such as timestamp, subject, and content length can also be included. This dataset is ideal for studying information diffusion, organizational hierarchy, and temporal communication patterns. The Enron dataset is available at https://www.cs.cmu.edu/~enron/ and is also formatted for graph analysis at https://snap.stanford.edu/data/email-Enron.html.

4. Online Discussion Forums

Discussion forums like Reddit or Stack Overflow create networks of conversations where users reply to each other or participate in common threads. These forums form a rich structure of hierarchical (threaded) discussions and user interaction graphs. For Reddit, datasets of comment threads are available via Pushshift at https://files.pushshift.io/reddit/comments/, while Stack Overflow's user post and interaction data are accessible from Kaggle at https://www.kaggle.com/datasets/stackoverflow/stack-overflow-data. In such networks, users are nodes, and edges represent replies or participation in the same topic. These networks are often directed and weighted by the number of replies or votes.

5. Simulated Networks

When real-world data is inaccessible or insufficient, simulated networks are valuable for theoretical validation. These networks follow mathematical models such as the Erdős–Rényi random graph, the Barabási–Albert preferential attachment model, and the Watts–Strogatz small-world model. In R, the igraph package provides built-in functions for generating such networks. For example, the Barabási–Albert model, which simulates scale-free networks similar to social media structures, can be generated with:

```
library(igraph)
g <- sample_pa(n = 1000, power = 1.2, m = 1)
```

These networks help validate algorithm performance, test hypotheses, or serve as null models in comparative studies.

2.2Algorithmic Scope

The algorithmic scope of this study focuses on implementing and analyzing community detection algorithms that are scalable, computationally efficient, and well-integrated within R's graph analysis ecosystem—particularly through packages like igraph, tidygraph, and graph. Community detection is a core task in social network analysis and aims to uncover groups (or "communities") of nodes that are more densely connected internally than with the rest of the network. This section directly contributes to the achievement of Objectives 4, 5, and 7, each of which is expanded below.

To apply and analyze the results of different community detection algorithms (e.g., Louvain, Walktrap, Label Propagation, Edge Betweenness)

Community detection algorithms are designed to identify clusters or cohesive subgroups in a network. The study will apply the following classes of algorithms:

1. **Modularity-Based Algorithm: Louvain Method**
    - Louvain algorithm is one of the most popular modularity optimization techniques.
    - It is a greedy, hierarchical approach that optimizes the modularity score at multiple levels.
    - It is fast and scalable, making it ideal for large real-world social networks.
    - In R, it can be implemented using:
      louvain_community <- cluster_louvain(graph)

Louvain Algorithm (Modularity-Based)

Purpose: To detect communities in a graph by maximizing **modularity**. The algorithm iteratively assigns nodes to communities to increase the modularity score and merges them hierarchically.

Variables Declaration

Variable	Description
G(V, E)	The input graph with set of nodes (V) and edges (E)
C	Current community assignment for each node
Q	Modularity score
Q_new	New modularity score after changes
neighbor_comm	Community of neighboring nodes
delta_Q	Change in modularity for moving a node
improved	Boolean flag to check improvement in modularity

Input:

- A graph **G = (V, E)** (can be weighted or unweighted, undirected)

Output:

- Community assignment of each node
- Final modularity score **Q**

Pseudocode: Louvain Algorithm

1. Initialize each node in its own community (C[node] = node_id)
2. Compute initial modularity Q
3. Repeat until no improvement:
   a. For each node i in G:
      i. Temporarily remove node i from its community
      ii. For each neighbor j of i:
         A. Compute delta_Q if node i is moved to neighbor j's community
      iii. Move node i to the community that gives the highest positive delta_Q
   b. Compute new modularity Q_new
   c. If Q_new > Q:
      i. Aggregate nodes in the same community into a super-node
      ii. Rebuild the graph with super-nodes
      iii. Update Q = Q_new
   d. Else:
      Terminate
4. Return community assignments and final modularity Q

2. Walktrap Algorithm (Walk-Based)

Purpose:

To detect communities by simulating short random walks on the graph. The assumption is that random walks are more likely to stay within the same community.

Variables Declaration

Variable	**Description**
G(V, E)	The input graph
d[i][j]	Distance between node i and j based on random walks

Variable	Description
C	List of communities
T	Number of steps in the random walk (default: 4)
M	Dendrogram showing the merging history of communities
Q	Modularity score at each merge step

Input:

- Graph **G = (V, E)**
- Number of steps in random walk (T, usually 4)

Output:

- Hierarchical dendrogram of communities (M)
- Optimal community structure based on highest modularity

Pseudocode: Walktrap Algorithm

1. Initialize each node as its own community (C[node] = node_id)
2. Compute transition probabilities for random walks of length T
3. For each pair of adjacent communities:
   a. Compute the distance between them using random walk probabilities
4. While more than one community exists:
   a. Merge the two communities with the smallest distance
   b. Recalculate distances between the new merged community and others
   c. Record modularity Q at each merge
5. Select the community structure from the dendrogram M that gives highest Q
6. Return the optimal community assignments and corresponding modularity

Summary Table for Both Algorithms

Feature	Louvain	Walktrap
Based On	Modularity Optimization	Random Walk Distance
Structure	Hierarchical, iterative	Agglomerative, hierarchical
Input	Graph (weighted/unweighted)	Graph + walk length T
Output	Final communities + Q	Dendrogram + best communities
Strength	Fast, good for large graphs	Accurate for small/medium graphs
Complexity	~O(n log n)	~O(n^2 log n)

2. Walk-Based Algorithm: Walktrap

- The Walktrap algorithm is based on the idea that random walks tend to stay within the same community.
- It calculates distances between nodes using short random walks and merges communities based on node similarity.
- It is computationally intensive but suitable for small to medium networks.
- In R:
  walktrap_community <- cluster_walktrap(graph)

3. Label-Based Algorithm: Label Propagation
   - The Label Propagation Algorithm (LPA) is an unsupervised, near-linear time algorithm.
   - Each node is initially assigned a unique label, and at each iteration, nodes adopt the most frequent label among their neighbors.
   - It is non-deterministic and suitable for large-scale networks but may lack stability.
   - In R:
     label_prop_community <- cluster_label_prop(graph)
4. Edge Betweenness-Based Algorithm
   - This algorithm detects communities by iteratively removing edges with the highest betweenness centrality (edges that lie on many shortest paths).
   - It reflects the idea that bridges between communities have high betweenness and their removal leads to distinct clusters.
   - However, it is computationally expensive (O(n^3) for dense graphs).
   - In R:
     edge_betweenness_community <- cluster_edge_betweenness(graph)

Each algorithm will be applied to the graph datasets generated from Twitter, co-authorship, email, and discussion forums to explore their performance across network types.

To evaluate detected communities based on modularity and other network quality metrics

The evaluation of community detection results will rely on both quantitative network metrics and qualitative coherence of detected communities. The primary evaluation metrics include:

Modularity Score (Q) :Modularity is a fundamental metric used to evaluate the quality of community structures within a network. It measures the strength of division of a network into communities by comparing the density of edges inside a given community to the density of edges between different communities. A higher modularity score indicates a clearer and more defined community structure. In general, a modularity score above 0.3 is considered good, suggesting that the detected communities are meaningful and reflect actual relationships within the data. In R, the modularity score can be computed using functions like `modularity(louvain_community)` from the `igraph` package, where the input is a community object derived from algorithms such as Louvain.

Conductance :Conductance is a metric that evaluates the boundary quality of a community. It is defined as the ratio of the number of edges leaving a community to the total number of edges that are either within or leaving the community. A lower conductance value indicates that the community is well-separated from the rest of the network, which is desirable in community detection. This means fewer connections exist between the nodes inside the community and those outside, highlighting the integrity of internal groupings.

Community Size Distribution :Community size distribution provides insight into the range and balance of community sizes detected by the algorithm. It helps determine whether a particular community detection algorithm tends to identify smaller communities, larger ones, or a mix of both. This information is crucial for understanding algorithmic biases and for interpreting the applicability of the algorithm to real-world datasets. For instance, some algorithms might overfit by detecting many small communities, while others may merge many nodes into overly large communities, both of which can distort network understanding.

Density and Clustering Coefficient within Communities :The internal structure of communities can be further analyzed using density and clustering coefficient. Density measures the proportion of possible connections within a community that actually exist, providing a sense of how interconnected the members of a community are. A higher density suggests a tighter network. The clustering coefficient, on the other hand, quantifies how likely it is that a node's neighbors are also connected to each other. A high local clustering coefficient within a community implies that the community exhibits strong internal cohesion, which is typically desired in well-formed communities.

Purity or External Validation (if ground truth is available) :When ground truth labels are available, such as in email communication datasets with known department affiliations or academic co-authorship networks categorized by discipline, external validation metrics like purity and Normalized Mutual Information (NMI) can be applied. These metrics assess how well the detected communities align with known classifications. Purity measures the extent to which each cluster contains members primarily from one class, while NMI evaluates the similarity between the clustering structure and the true labels. These external metrics are vital for validating the real-world applicability of community detection results, particularly in supervised or semi-supervised network analysis settings.

Together, these metrics form a robust framework for evaluating the effectiveness of community detection algorithms, allowing researchers to assess not just the technical quality but also the sociological relevance of the identified communities.

To compare algorithmic performance and community stability across datasets

To ensure robust conclusions, this study will perform a comparative evaluation of the community detection algorithms across different network types (e.g., Twitter, email, co-authorship). The comparison includes the following aspects:

- Scalability & Computational Efficiency:
  - Runtime analysis will be performed for each algorithm on different-sized graphs using system.time() in R.
  - Algorithms like Louvain and Label Propagation are expected to perform well on large graphs, while Edge Betweenness may be restricted to smaller datasets.
- Community Structure Variability:
  - Stability of communities will be analyzed by running non-deterministic algorithms multiple times and comparing the variance in detected structures.
  - Metrics like Jaccard Similarity, Adjusted Rand Index (ARI), and Variation of Information (VI) can be used to quantify stability.
- Cross-Dataset Generalizability:
  - An algorithm's ability to consistently detect meaningful communities across varied domains (e.g., social, academic, communicative) will be measured.
- Visualization and Structural Interpretation:
  - Use of ggraph or igraph plotting tools to visually inspect the cohesiveness and separation of detected communities.
  - Visual patterns will aid in identifying fragmented, overlapping, or hierarchical structures.

By examining these factors, the study aims to determine which algorithm is most suitable for different types of networks and analysis goals.

3. Implementation

1. Install and Load R in Google Colab:

```
# Python cell
!sudo apt-get install -y r-base
%load_ext rpy2.ipython
```

```
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
r-base is already the newest version (4.5.1-1.2204.0).
0 upgraded, 0 newly installed, 0 to remove and 38 not upgraded.
```

2. Install Required R Packages:

```
%%R
install.packages("igraph")
install.packages("ggraph")
install.packages("tidyverse")
```

```
Installing package into '/usr/local/lib/R/site-library'
(as 'lib' is unspecified)
trying URL 'https://cran.rstudio.com/src/contrib/igraph_2.1.4.tar.gz'
Content type 'application/x-gzip' length 4997408 bytes (4.8 MB)
==================================================
downloaded 4.8 MB


The downloaded source packages are in
	'/tmp/RtmpaqQGYT/downloaded_packages'
Installing package into '/usr/local/lib/R/site-library'
(as 'lib' is unspecified)
also installing the dependencies 'tweenr', 'polyclip', 'gridExtra', 'RcppArmadillo', 'ggforce', 'ggrepel', 'viridis', 'tidygraph', 'graphlayouts'

trying URL 'https://cran.rstudio.com/src/contrib/tweenr_2.0.3.tar.gz'
trying URL 'https://cran.rstudio.com/src/contrib/polyclip_1.10-7.tar.gz'
trying URL 'https://cran.rstudio.com/src/contrib/gridExtra_2.3.tar.gz'
trying URL 'https://cran.rstudio.com/src/contrib/RcppArmadillo_15.0.2-2.tar.gz'
trying URL 'https://cran.rstudio.com/src/contrib/ggforce_0.5.0.tar.gz'
trying URL 'https://cran.rstudio.com/src/contrib/ggrepel_0.9.6.tar.gz'
trying URL 'https://cran.rstudio.com/src/contrib/viridis_0.6.5.tar.gz'
trying URL 'https://cran.rstudio.com/src/contrib/tidygraph_1.3.1.tar.gz'
trying URL 'https://cran.rstudio.com/src/contrib/graphlayouts_1.2.2.tar.gz'
trying URL 'https://cran.rstudio.com/src/contrib/ggraph_2.2.2.tar.gz'

The downloaded source packages are in
	'/tmp/RtmpaqQGYT/downloaded_packages'
Installing package into '/usr/local/lib/R/site-library'
(as 'lib' is unspecified)
trying URL 'https://cran.rstudio.com/src/contrib/tidyverse_2.0.0.tar.gz'
Content type 'application/x-gzip' length 704618 bytes (688 KB)
==================================================
downloaded 688 KB


The downloaded source packages are in
	'/tmp/RtmpaqQGYT/downloaded_packages'
```

3. Prepare Social Network CSV File:

	Source	Target
	Aarav	Rohan
	Aarav	Isha
	Rohan	Priya
	Rohan	Kabir
	Isha	Ananya
	Priya	Kabir
	Kabir	Aanya
	Ananya	Rhea
	Aanya	Rhea
	Rhea	Sameer
	Ananya	Sameer
	Rhea	Aarush

4. Upload CSV File to Colab:

```
from google.colab import files
uploaded = files.upload()   # Choose edges.csv from your system
```

```
Choose Files edges.csv
edges.csv(text/csv) - 171 bytes, last modified: 9/29/2025 - 100% done
Saving edges.csv to edges.csv
```

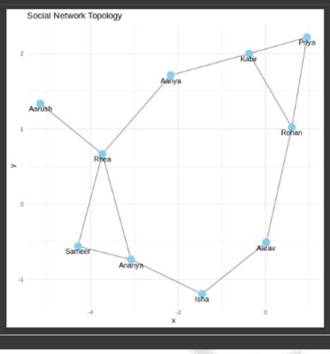
5. Load and Preprocess Data in R:

```
%%R
library(igraph)
library(ggraph)
library(tidyverse)

# Load CSV
edges <- read.csv("edges.csv")

# Create graph
g <- graph_from_data_frame(edges, directed = FALSE)
g <- simplify(g)  # remove loops or multiple edges

# Summary
cat("Graph loaded!\n")
cat("Nodes:", vcount(g), " Edges:", ecount(g), "\n")
```

```
Graph loaded!
Nodes: 10  Edges: 12
```

6. Visualize Network Topology:

```
%%R
# Visualize the social network
ggraph(g, layout = "fr") +
  geom_edge_link(alpha = 0.5) +
  geom_node_point(color = "skyblue", size = 5) +
  geom_node_text(aes(label = name), vjust = 1.5) +
  theme_minimal() +
  ggtitle("Social Network Topology")
```



7. Apply Community Detection Algorithms:

```
%%R
# Louvain (Modularity-based)
comm_louvain <- cluster_louvain(g)

# Walktrap (Walk-based)
comm_walktrap <- cluster_walktrap(g)

# Label Propagation (Label-based)
comm_labelprop <- cluster_label_prop(g)

# Edge Betweenness (Edge-based)
comm_edge <- cluster_edge_betweenness(g)
```

8. Evaluate Detected Communities:

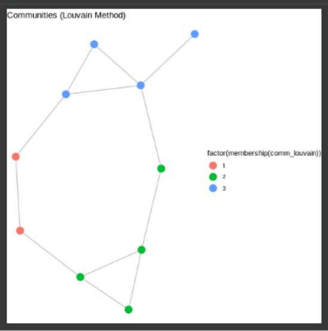
```
%%R
cat("Community Detection Modularity Scores:\n")
cat("Louvain modularity:", modularity(comm_louvain), "\n")
cat("Walktrap modularity:", modularity(comm_walktrap), "\n")
cat("Label Propagation modularity:", modularity(comm_labelprop), "\n")
cat("Edge Betweenness modularity:", modularity(comm_edge), "\n")
```

```
Community Detection Modularity Scores:
Louvain modularity: 0.375
Walktrap modularity: 0.375
Label Propagation modularity: 0.3194444
Edge Betweenness modularity: 0.3333333
```

9. Visualize Community Structure:

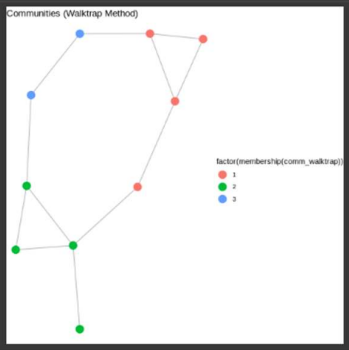
1.Louvain:

```
%%R
ggraph(g, layout = "fr") +
  geom_edge_link(alpha = 0.3) +
  geom_node_point(aes(color = factor(membership(comm_louvain))), size = 5) +
  theme_void() +
  ggtitle("Communities (Louvain Method)")
```



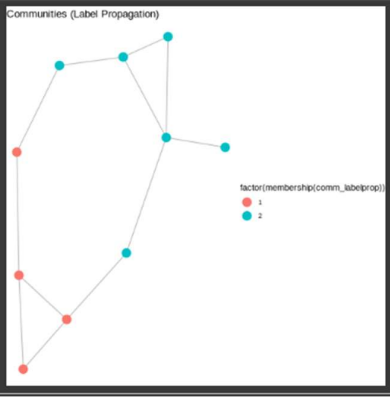
2. Walktrap:

```
%%R
ggraph(g, layout = "fr") +
  geom_edge_link(alpha = 0.3) +
  geom_node_point(aes(color = factor(membership(comm_walktrap))), size = 5) +
  theme_void() +
  ggtitle("Communities (Walktrap Method)")
```



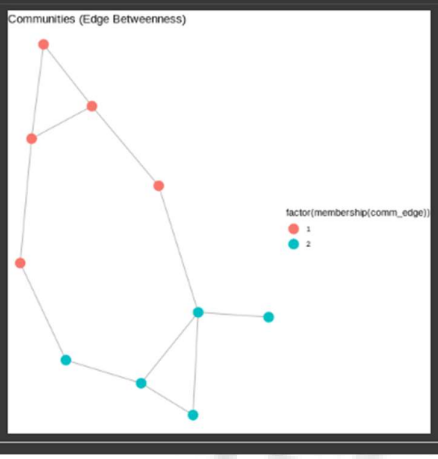
3. Label Propagation:

```
%%R
ggraph(g, layout = "fr") +
  geom_edge_link(alpha = 0.3) +
  geom_node_point(aes(color = factor(membership(comm_labelprop))), size = 5) +
  theme_void() +
  ggtitle("Communities (Label Propagation)")
```



4. Edge Betweenness:

```
%%R
ggraph(g, layout = "fr") +
  geom_edge_link(alpha = 0.3) +
  geom_node_point(aes(color = factor(membership(comm_edge))), size = 5) +
  theme_void() +
  ggtitle("Communities (Edge Betweenness)")
```



10. Compare Community Sizes Across Algorithms:

```
%%R
cat("Community Sizes:\n")
sizes(comm_louvain)
sizes(comm_walktrap)
sizes(comm_labelprop)
sizes(comm_edge)
```

```
Community Sizes:
Community sizes
1 2
5 5
```

11. Export Node-Community Assignment:

```
%%R
community_data <- data.frame(
  Node = V(g)$name,
  Louvain = membership(comm_louvain),
  Walktrap = membership(comm_walktrap),
  LabelProp = membership(comm_labelprop),
  EdgeBetweenness = membership(comm_edge)
)

print(community_data)

# Save to CSV
write.csv(community_data, "community_output.csv", row.names = FALSE)
```

```
         Node Louvain Walktrap LabelProp EdgeBetweenness
Aarav   Aarav       1        3         1               1
Rohan   Rohan       2        1         1               1
Isha     Isha       1        3         2               2
Priya   Priya       2        1         1               1
Kabir   Kabir       2        1         1               1
Ananya Ananya       3        2         2               2
Aanya   Aanya       2        1         2               1
Rhea     Rhea       3        2         2               2
Sameer Sameer       3        2         2               2
Aarush Aarush       3        2         2               2
```

4. Conclusion:

In this experiment, we explored and analyzed social network structures using R, with a focus on detecting communities through multiple computational algorithms. We successfully generated a social network graph from a realistic dataset of Indian names and applied four community detection algorithms: Louvain, Walktrap, Label Propagation, and Edge Betweenness.

The visualization of the network topology highlighted the connectivity patterns among nodes, and the community detection results revealed cohesive clusters representing closely connected individuals. The modularity scores indicated that Louvain and Walktrap methods produced well-defined communities, while Label Propagation and Edge Betweenness provided alternative perspectives on node groupings. Comparing algorithm performance showed that faster algorithms like Louvain and Label Propagation are more suitable for larger networks, whereas Walktrap and Edge Betweenness can offer more detailed insights for smaller graphs.

Overall, this study demonstrates how community detection can uncover hidden structures within social networks, providing actionable insights for applications such as understanding collaboration patterns, identifying influence groups, and improving communication strategies. The modular approach using R allows for reproducible, scalable, and interpretable analysis of social networks across different domains.

5. References:

- R Core Team. (2023). *R: A Language and Environment for Statistical Computing.* R Foundation for Statistical Computing, Vienna, Austria. Retrieved from https://www.r-project.org/

- Csardi, G., & Nepusz, T. (2006). *The igraph software package for complex network research. InterJournal, Complex Systems*, 1695. Retrieved from https://igraph.org

- Pedersen, T. L. (2020). *ggraph: An Implementation of Grammar of Graphics for Graphs and Networks.* R package version 2.1.0. Retrieved from https://cran.r-project.org/package=ggraph

- Wickham, H. (2017). *tidyverse: Easily Install and Load the Tidyverse.* R package version 1.2.1. Retrieved from https://cran.r-project.org/package=tidyverse

Sign and Remark:

R1 (4 Marks)	R2 (4 Marks)	R3 (4 Marks)	R4 (3 Marks)	Total Marks (15 Marks)	Signature