DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

NATIONAL INSTITUTE OF TECHNOLOGY SILCHAR

SOCIAL NETWORK & ANALYSIS

MINI PROJECT

A Report on disjoint community detection in dynamic networks

Group 1

Mayukh Saikia	-	2012030 (Team Lead)
Anurag Pathak	-	2012018
Angshuman Acharya	-	2012044
Uday Sankar Das	-	2012047
Jyotirmoy Das	-	2012054

Submitted to:

Dr. Anupam Biswas

ABSTRACT

Detecting communities in dynamic evolving networks is of great interest. It has received tremendous attention from researchers. One promising solution is to update communities incrementally taking the historical information into consideration. However, most of the existing methods are only suitable for the case of one node adding or one edge adding. Factually, new data are always generated continuously with subgraphs joining simultaneously in dynamic evolving networks. To address the above problem, we present an incremental method to detect communities by handling subgraphs. We first make a comprehensive analysis and propose four types of incremental elements. Then we propose different updating strategies. Finally, we present the algorithms to detect communities incrementally in dynamic evolving networks. The experimental results on real-world data sets indicate that the proposed method is effective and has superior performance compared with several widely used methods.

A vast majority of existing algorithms, however, treats temporal networks as a collection of snapshots, thus struggling with stability and continuity of detected communities. Inspired by an observation that similarly behaving agents tend to self-organize into the same cluster during epidemic spreading, we devised a novel community detection approach for temporal networks based on a susceptible-infectious-recovered-like (SIR-like) spreading process. To account for network time evolution, we used communities from the preceding time step to modulate spreading in the current time step. The approach is based on a combination of network clustering and dynamic network analysis techniques. The we demonstrate the effectiveness of their method on several real-world temporal networks and compare it to existing state-of-the-art methods.

CONTENTS

Chapter 1

INTRODUCTION

The fast development of information technology has accelerated the prevalence of social media, where people are able to share news, pictures, publish opinions and comments anytime anywhere. All this data, together with users' behaviors and social relations, have abundant potential values. As a result, social network analysis has become a hot research area and has received considerable attention

Community detection aims to identify cohesive clusters or groups in real-world graphs such as social networks, web graphs and biological networks. It is a problem of considerable practical interest. Identifying user communities with similar interests can foster sharing resources; detecting consumer communities from online shopping sites is beneficial to target potential customers; discovering student communities from online education systems can promote the collaborative learning efficiency of students. Thus, it has received a great deal of attention [8,9]. Numerous techniques have been developed for community detection, such as partition-based methods, hierarchical clustering methods, density-based methods, modularity optimizing methods and game theory-based methods.

In view of some limitations of existing methods, we characterize the community detection in dynamic evolving networks with tools that enable to support the following applications. The method should be capable of handling subgraphs (including nodes and edges) addition. Moreover, the communities should be identified by taking the historical influence into account.

To answer the above questions, in this paper, we investigate: how to model and analyze the incremental elements comprehensively, and how to update communities incrementally taking into account historical information with no human intervention. In an attempt to address the above problems, we propose a new incremental method to identify communities in dynamic evolving social networks. Our research contributions are summarized as follows

- provide a comprehensive analysis on the incremental elements and propose four different types;
- present the incremental updating strategies for the different types of incremental elements;
- propose an incremental algorithm to detect communities in dynamic evolving social network;
- evaluate the proposed algorithm extensively with real-world social networking data;

Community detection has become a topic of great importance and interest in network science. Heretofore, however, most algorithms for the purpose of community detection focused on static networks whereas, in practice, system topology often evolves in time. We propose a novel approach for detecting communities in temporal networks using a spreading process. We took this proposed algorithm and implemented the algorithm proposed in the paper, along with a detailed description of the methodology used. We will also discuss the results obtained from applying the algorithm on different datasets and compare them with existing community detection algorithms. Tried to reform and give our take on the process. Some problems were first recognized in the context of *clustering algorithms*, leading to an approach dubbed evolutionary clustering.

Here, we exploit this fact by first using a dynamical process to extract structural information. Subsequently, we adjust the dynamics to reflect network topology from the previous time step while searching for new structural information in the current time step. We then assessed the performance of this spreading-based similarity measure for the purpose of community detection, especially in situations where network topology evolves in time. The co-evolution of spreading dynamics and topology, in particular, enabled robust community detection in temporal networks such that wild community fluctuations from one moment to another are avoided.

Chapter 2

SOME BASIC CONCEPTS

This section formulates the research problem by introducing some definitions.

Definition 1 :(Network Stream (GS)). We define the graph stream (GS) as a sequence of networks at different time slices: GS = {G^1, G^2, ……… G^t, ……}, where
- G^t, represents the snapshot of network at time t, and G^t = (V(G^t), E(G^t));
- V(G^t) denotes the set of vertices in G^tt ;
- E(G^t) denotes the set of edges in G^t

Definition 2 : (Community Structure (CS^t)). Let CS^t represent the set of community structures detected from G^t . That is, CS^t = {C_1^t, C_2^t, …… C_i^t, …… $C_{K^t}^t$ }, where
- K^t represents the number of communities in G^t;
- C_i^t represents the ith community of G^t, i = 1, 2, 3, . . . , K;
- $C_k^t \cap C_l^t$is not necessary to be empty;

Definition 3 : (Community Stream (CS)). We define the community stream as CS = {CS^1, CS^2, ,…..CS^t, . . .}, where CS^t denotes set of community structures detected from network G^t.

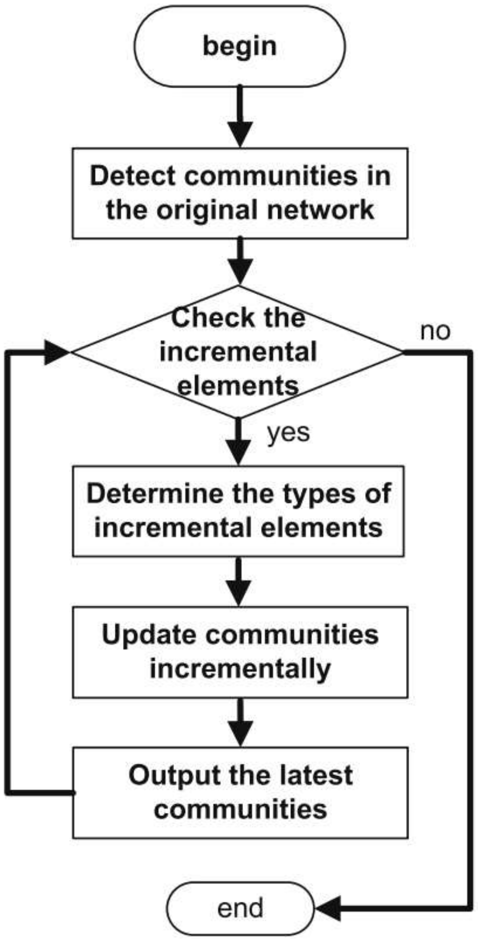

begin
Detect communities in the original network
Check the incremental elements
no
yes
Determine the types of incremental elements
Update communities incrementally
Output the latest communities
end

Fig. The framework to detect communities incrementally in evolving social networks.

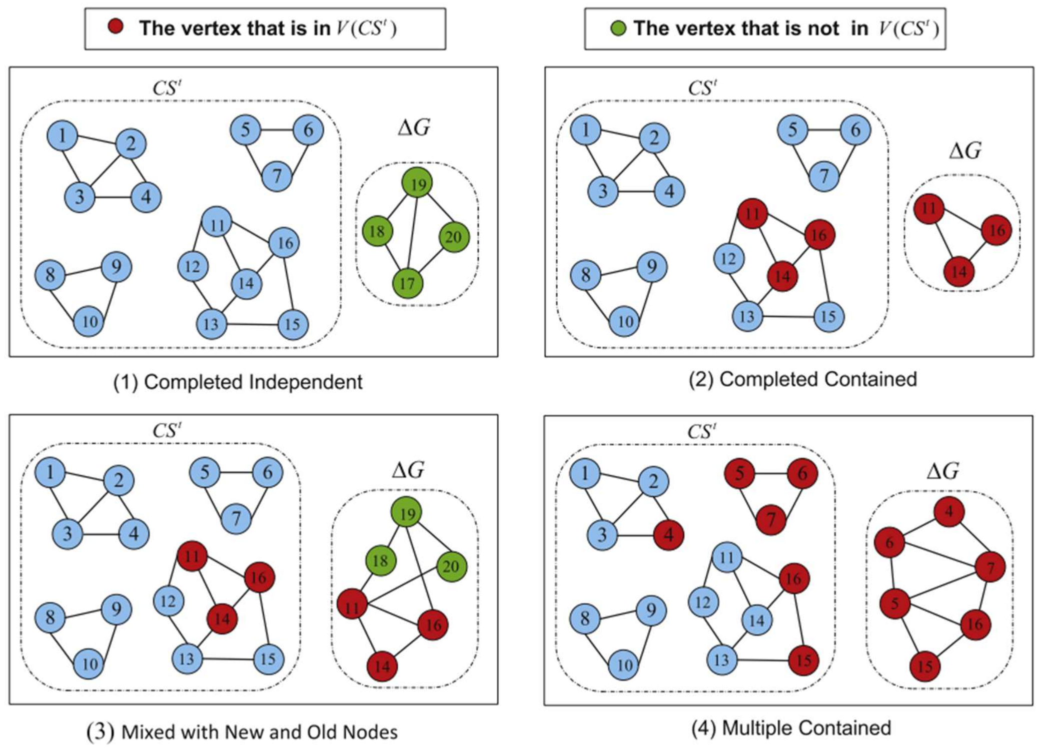
Proposed method

- **The framework to detect communities in dynamic networks:**

  The communities should be first identified in the original social network at time t0. As time passes, it is required to collect the incremental elements in the social network and make a formal analysis. Then the incremental updating strategies are designed based on the communities at the last state. Finally, the latest communities are derived, and the evolutionary paths are also updated accordingly

- **The analysis of incremental elements:**

  The incremental elements ∆G t+1 refer to all changes between time t and t+1, which could be composed of various subgraphs. Let subG represent one subgraph of ∆G t+1 , then V(subG) denotes the set of vertices in subG and E(subG) denotes the set of edges in subG. According to the relationship between subG and the communities ing social networks is illustrated in Fig. 2. The communities should be first identified in the original social network at time t0. As time passes, it is required to collect the incremental elements in the social network and make a formal analysis. Then the incremental updating strategies are designed based on the communities at the last state. Finally, the latest communities are derived, and the evolutionary paths are also updated accordingly. 4.2. The analysis of incremental elements The incremental elements ∆G t+1 refer to all changes between time t and t+1, which could be composed of various subgraphs. Let subG represent one subgraph of ∆G t+1 , then V(subG) denotes the set of vertices in subG and E(subG) denotes the set of edges in subG. According to the relationship between subG and the communities Fig. 2. The framework to detect communities incrementally in evolving social networks. at previous time slice CSt , we propose the following four types of subG.


● The vertex that is in $V(CS^t)$
● The vertex that is not in $V(CS^t)$
CS^t
ΔG
(1) Completed Independent
(2) Completed Contained
(3) Mixed with New and Old Nodes
(4) Multiple Contained

(1) Complete Independent (CompnIde)

The incremental subgraph subG is considered to be complete independent iff none of its vertices is in a community of CS^t . That is, all the vertices of subG are newly joined in the network at time t+1. The complete independent incremental subgraph type is defined as follows:

CompInde(subG) = 1 iff

$(\forall C_k^t)(C_k^t \in CS^t) \rightarrow (V(subG) \cap V(C_k^t) = \emptyset)$

(2) Complete Contained(CompCont)

The incremental subgraph subG is considered to be complete contained iff its vertices have appeared before in a certain community of CS^t . Hence, all those vertices in subG can be considered as old nodes. The complete contained incremental subgraph type is formally defined as follows:

CompCont(subG) = 1 iff

$(\forall v)$ $(v \in V(subG) \rightarrow (\exists C_k^t)v \in V(C_k^t)$

(3) Mixed with New and Old Nodes (Mixed)

The incremental subgraph subG is considered to be mixed with new and old nodes iff it has old nodes and new nodes in subG. This means that in subG there are some (old) vertices belonging to one or several communities of CSt and other new emerging vertices joining the network at time t +1. The formal definition of this type of incremental subgraph is given as follows:

Mixed(subG) = 1 iff

$(\exists C_k^t)(V(C_k^t) \cap V(subG) \not\models \emptyset) \wedge$

$(\exists v)(v \in V(subG) \wedge v \,/\!\in V(CS^t))$

(4) Multiple Contained (MultiCont)

The incremental subgraph subG is considered to be multiple contained iff all its vertices are old ones distributed in several (more than one) communities of CS^t . The formal definition of this type of incremental subgraph is given as follows:

MultiCont(subG) = 1 iff

$(\forall v)v \in V(subG) \rightarrow (v \in V CS^t))$

$\wedge(\exists C_k^t)(\exists C_i^t)(C_k^t \cap subG \not\models \emptyset)$

$\wedge(C_i^t \cap subG \not\models \emptyset)$

Chapter 3.1 (Paper-1)

ALGORITHMS AND WORKING

Algorithm 1: Update communities for the Mixed type.
Input : V_{old}, V_{new}, C_k^t, subG
Output: the communities updated at t+1

```
1.  Update$CS^{t+1}$ $\longleftarrow$ Ø;
2.  for each v ∈ $V_{old}$ do                    // update old nodes first
3.          calculate $S_v^{C_k^t}$, $S_v^{subG}$ according to Equ. (9);
4.      if $S_v^{C_k^t} \leq S_v^{subG}$ then
5.              copy v to subG;
6.      else
7.              remove v from subG and add it to $C_k^t$ ;
8.      end if
9.  end for
10. for each v ∈ $V_{new}$ do                    // update new nodes next
11.     calculate $S_v^{C_k^t}$, $S_v^{subG}$ according to Equ. (9);
12.     if $S_v^{C_k^t} \leq S_v^{subG}$ then
13.             copy v to subG;
14.     else
15.             remove v from subG and add it to $C_k^t$;
16.     end if
17. end for
18. Update$\longleftarrow$ $CS^{t+1}$        update $CS^{t+1}$U (V(subG)) ∪ {V( $C_k^t$)};
19. return Update $CS^{t+1}$;
```

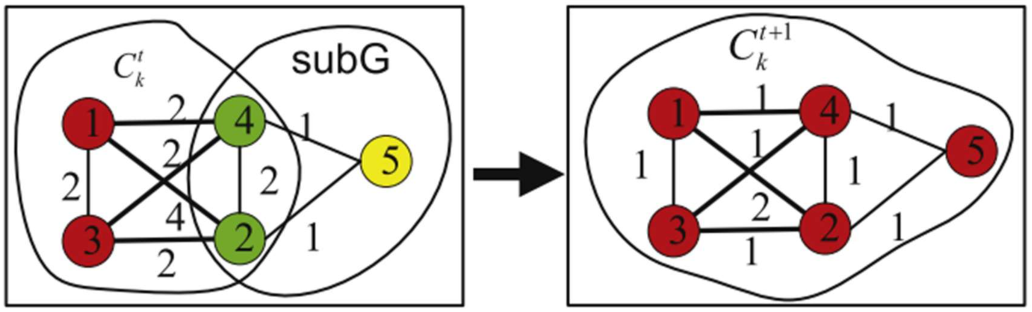


Fig. Community updating for the type of Mixed: the enlarging of the previous community.

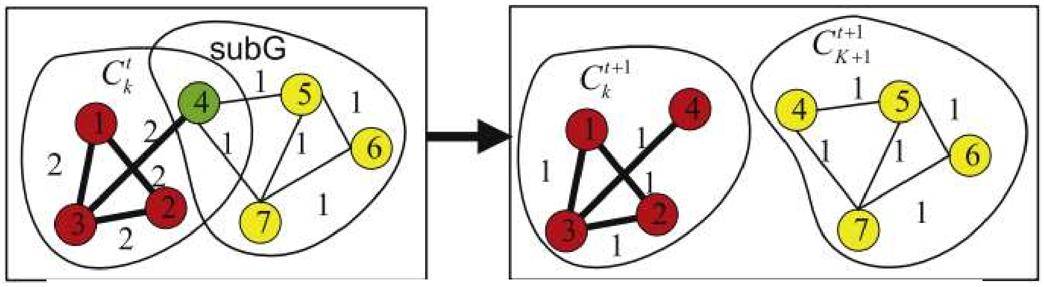


Fig. Community updating for the type of Mixed: the birth of new community

Algorithm 2 : Update communities for the MultiCont type
Input: C t 1 , C t 2 , ..., C t K , subG
Output: the communities updated at t + 1

```
1.  Update$CS^{t+1}$ ⟵ Ø;
2.  for k = 1; k ≤ K; k ++  do
3.          $V_k$ ⟵ $V(C_k^t)$ ∩ subG;
4.  for each v ∈ $V_k$ do
                    calculate $S_v^{C_k^t}$ , $S_v^{subG}$ according to Equ. (9);
5.  end for
6.  end for
7.  for each v in U $V_k$ do
8.  if max($S_v^{C_k^t}$ ) ≤ $S_v^{subG}$ then
                copy v to subG;
9.  else
                remove v from subG and add it to the community with maximum
                strength;
10. end if
11. end for
12. update$CS^{t+1}$ ⟵ update$CS^{t+1}$ ∪ {V(subG)) U {V($C_k^t$)};
13. return update$CS^{t+1}$ ;
```

It is necessary to mention that the algorithm is used outside of the network. In our work, we did not design any mechanism for finding the changes. The changes are detected by checking the time stamps. For instance, one person sends an email to another, this social event will take the time stamp in nature with the email service. In the microblogging platform, the time when a user uploads/replies/forwards some postings could be recorded automatically. Thus, when we design and implement the crawling program, we can get the timestamp of the users (nodes) and social relationship (edge) easily. With the help of timestamp, we can get the changes (the new coming nodes and edges).

The types of those incremental elements at hand, we can update communities incrementally. The algorithm to detect communities incrementally in time-evolving network is presented in Algorithm 4. The running time of the step 2 is O(n). The most time-consuming parts of the Algorithm 4 are from step 4 to step 20. The time complexity of the above parts is O(S × K × L), where S denotes the number of incremental subgraphs; K denotes the number of communities at t and it is nearly a constant; L = max{|V(C t j)|, |V(subG)|} and L ≪ n. The running time of step 21 to step 25 is O(E), where E = max{|E(C t j)|, |E(subG)|}. Thus, the time complexity of Algorithm 4 is O(n + S × K × L + E). That is, the proposed incremental method has a nearly linear complexity, which guarantees high efficiency.

Chapter 3.2 (Paper-2)

ALGORITHMS AND WORKING

We are given a set of time steps {1, 2, ……., T}, a temporal network is a set of sequential graphs , G = {$\boldsymbol{G}^{[1]}$, $\boldsymbol{G}^{[2]}$, … … … … , $\boldsymbol{G}^{[T]}$ }, where $\boldsymbol{G}^{[t]}$ = (V, $\boldsymbol{E}^{[t]}$) and V = {v1, v2,………., vN} is a set of nodes while $\boldsymbol{E}^{[t]} \subseteq \mathbf{V} \times \mathbf{V}$ is a set of links between nodes at time t. An alternative way of representing $\boldsymbol{G}^{[t]}$ is by N × N adjacency matrix, A^[t].

Our implementation of the algorithm for community detection comprises mainly following steps:

1. A spreading process to quantify similarity between any pair of nodes in the network,
2. A greedy clustering algorithm to partition the network into communities,
3. An extension to account for the network's temporal evolution in an effective manner.

a) Spreading: We adopted a traditional SIR model in which each individual might be in one of the following three states: susceptible S, infectious I, or recovered R. S individuals may get infected due to the physical contacts with their I neighbors, and this occurs with infection probability λ. Furthermore, I individuals may recover with probability μ. Given an initial number of I individuals, the spreading process has a natural end when there are no more I individuals left in the network. Our interest, however, is not necessarily in this final state, but rather states that best quantify node similarity.

The following transition probability characterizes the Markov model for SIR spreading process:

$$p_i^{[t]}(S,\tau+1) = p_i^{[t]}(S,\tau)\prod_{j=1}^{N}(1-\lambda A_{ij}^{[t]}p_j^{[t]}(I,\tau))$$

$$p_i^{[t]}(I,\tau+1) = p_i^{[t]}(S,\tau) \times\left[1-\prod_{j=1}^{N}(1-\lambda A_{ij}^{[t]}p_j^{[t]}(I,\tau))\right] + (1-\mu)p_i(I,t),$$

$$p_i^{[t]}(R,\tau+1) = \mu p_i^{[t]}(I,\tau) + p_i^{[t]}(R,\tau),$$

To characterize nodes based on the spreading process, we ran N simulations, where in each simulation different node vj is selected as the origin of the infection. The probability obtained from equation after running the simulation of the spreading process for T steps with node vj being the origin of the infection.

b) Clustering: Upon characterizing every network node vi with corresponding vector $P_i^{[t]}$, we turned to clustering, first by collecting all the information into an N × N partitioning matrix, $P^{[t]}$, at time t whose elements are:

$$P_{ij}^{[t]} = \begin{cases} \mathbf{p}_i^{[t]}(j), & i \neq j, \\ \frac{1}{N-1}\sum_{j\neq i} \mathbf{p}_i^{[t]}(j), & i = j. \end{cases}$$

Next, we quantified the node similarity by means of the Pearson correlation:

$$\mathrm{Cor}_{\mathrm{Pearson}}(v_i, v_j) = \frac{1}{N}\sum_{k=1}^{N} \frac{(P_{ik}^{[t]} - \mu_i^{[t]})(P_{jk}^{[t]} - \mu_j^{[t]})}{\sigma_i^{[t]}\sigma_j^{[t]}}$$

Where,

$$\mu_i^{[t]} = \frac{1}{N}\sum_{k=1}^{N} P_{ik}^{[t]},$$

$$\sigma_i^{[t]} = \sqrt{\frac{1}{N}\sum_{k=1}^{N}(P_{ik}^{[t]} - \mu_i^{[t]})^2}.$$

c) Generalization to temporal networks: Aiming to apply our proposed method to the investigation of temporal networks, we started by calculating the first partitioning matrix, P[1], and then performing clustering as described above to obtain community structure C[1]. If the consecutive realizations of a temporal network were independent, then we could treat the whole temporal network as a set of static networks, and proceed by finding any P[t] independently of any other P[s]. therefore, we took C[t−1] and G[t] as inputs and modified the spreading process to more likely infect nodes from within the same community than nodes between communities.

Chapter 3.3

Preprocessing of Enron email dataset:

We needed to process the Enron Email dataset. It was a 1GB+ file which contained all the email data. We wrote a code to achieve this.

- It imports the necessary libraries: csv, re, pandas, and shutil.
- It defines a function called extract() that extracts the month, year, sender, recipient, and cc recipients from an email header using regular expressions.
- It creates two folders to store the intermediate and final datasets.
- It reads a large CSV file called emails.csv in chunks and processes only those emails that are in the "sent" folder. It calls the extract() function to extract relevant information from the email header, such as the sender and recipient email addresses, and creates a list of unique senders.
- It reads the same CSV file again in chunks and processes only those emails that are in the "sent" folder. It calls the extract() function to extract relevant information from the email header and creates a list of tuples consisting of the sender, recipient, month, and year. If the recipient is also a sender, it creates an edge between them.
- It writes the extracted data into a CSV file in the intermediate folder called dataset-preprocessed-stage1.csv.
- It creates a mapping of email addresses to node IDs and writes it into a CSV file in the intermediate folder called email_to_id.csv.
- It creates a dictionary called file_to_data that maps filenames to a dictionary that maps edges to their weight.
- It reads the CSV file called dataset-preprocessed-stage1.csv and for each tuple, it computes the filename from the month and year and adds the edge to the corresponding dictionary.
- It writes the edges with their weight to separate CSV files for each month and year in the real-world-datasets folder.
- It prints the dictionary file_to_data containing the mapping of filenames to edges with their weight.

Chapter 4

EXPERIMENT AND RESULTS

Enron Email

Timestamp	Measure	Algo1	Algo2
t0	Modularity	0.63	0.65
	Conductance	0.29	0.27
	Expansion	0.45	0.56
	Cut Ratio	0.03	0.04
t3	Modularity	0.68	0.65
	Conductance	0.38	0.27
	Expansion	4.37	4.56
	Cut Ratio	0.04	0.04

College Msg

Timestamp	Measure	Algo1	Algo2
t0	Modularity	0.75	0.68
	Conductance	0.34	0.30
	Expansion	0.43	0.46

	Cut Ratio	0.12	0.10
t3	Modularity	0.68	0.65
	Conductance	0.36	0.37
	Expansion	3.67	3.64
	Cut Ratio	0.10	0.09

Synthetic

Timestamp	Measure	Algo1	Algo2
t0	Modularity	0.94	0.96
	Conductance	0.45	0.46
	Expansion	0.67	0.63
	Cut Ratio	0.23	0.18
	NMI	0.93	0.94
	ARI	0.78	0.75
	AMI	0.75	0.72
t3	Modularity	0.76	0.73
	Conductance	0.35	0.38

	Expansion	0.45	0.41
	Cut Ratio	0.34	0.31

Chapter 5

IMPROVISED ALGORITHM

We have tried to improve the spreading process model. We have observed that for the spreading process algorithm, it takes a lot of time to compute the communities. so, we have tried to reduce the time required for the calculation of the number of communities. We have achieved this by using an improvised version of Louvain algorithm to detect an approximate value of the best number of clusters. We then revalidate based on nearby integer values and find the best number of partitions. The following methods remain same.
Also, we made an attempt to get incremental communities in a less time-consuming manner. We have identified the incremental elements. Then we create a subgraph of only those communities which are nearby to the community which have changes made. Finally, we do the process on only the subgraph which reduces the time complexity.

Chapter 6

CONCLUSION

One of the main benefits of community detection is its ability to identify groups or clusters within a network that may have different roles or functions. Community detection is a crucial task in network analysis as it allows for the identification of groups or clusters of nodes that have similar characteristics or functions within a network. The research paper we implemented proposed a method for community detection based on modularity optimization, which is a widely used approach in the field. The method involves partitioning the network into groups that maximize the modularity score, which measures the degree of segregation between communities in a network.

Our implementation involved using the Python programming language and various network analysis libraries, such as NetworkX and Matplotlib. We applied the method to a real-world Enron Email dataset and were able to identify meaningful communities within the network. We also conducted various experiments to evaluate the effectiveness using a few synthetic datasets and compared it to other state-of-the-art community detection algorithms.

Overall, our implementation of the research paper on community detection provided us with a deeper understanding of the topic and the challenges involved in network analysis. We were able to apply the method to a real-world dataset and gain insights into the underlying structure of the network. Additionally, we were able to evaluate the effectiveness of the method and compare it to other approaches.

This information of community can be useful in a variety of applications, such as social network analysis, recommendation systems, and targeted advertising. Additionally, community detection can help in understanding the spread of diseases or information within a network, and identifying influential nodes or groups that can be targeted for intervention.

Despite the benefits of community detection, there are still many challenges in the field, such as the scalability of the algorithms to larger networks, the resolution limit problem, and the lack of ground-truth communities for evaluation. Future research can focus on addressing these challenges and developing more accurate and efficient methods for community detection.

In conclusion, implementing the research paper on community detection provided us with a valuable learning experience and insights into the field of network analysis. We were able to apply the method to a real-world dataset and gain insights into the structure of the network. Community detection has the potential to provide valuable information in various applications and future research can focus on addressing the challenges in the field to improve its effectiveness and scalability.

Chapter 7

REFERENCES

1. W. Wu, J. Zhao, C. Zhang, F. Meng, Z. Zhang, Y. Zhang, Q. Sun, Improving performance of tensor-based context-aware recommenders using bias tensor factorization with context feature auto-encoding, Knowl.-Based Syst. 128 (C) (2017) 71–77.

2. R. Cazabet, F. Amblard, C. Hanachi, Detection of overlapping communities in dynamical social networks, in: IEEE Second International Conference on Social Computing, 2010, pp. 309–314.

3. J. Xie, M. Chen, B.K. Szymanski, Labelrankt: incremental community detection in dynamic networks via label propagation, in: The Workshop on Dynamic Networks Management and Mining, 2013, pp. 25–32

4. Z. Zhao, W. Liu, Y. Qian, L. Nie, Y. Yin, Y. Zhang, identifying advisor-advisee relationships from co-author networks via a novel deep model, Inform. Sci. 466 (2018) 258–269

5. H.S. Cheraghchi, A. Zakerolhosseini, Toward a novel art inspired incremental community mining algorithm in dynamic social network, Appl. Intell. 46 (2) (2017) 409–426.

6. Z. Bu, J. Cao, H.J. Li, G. Gao, H. Tao, Gleam: a graph clustering framework based on potential game optimization for large-scale social networks, Knowl. Inf. Syst. 55 (3) (2018) 741–770.

7. Greene D., Doyle D. and Cunningham P., Tracking the Evolution of Communities in Dynamic Social Networks, in 2010 International Conference on Advances in Social Networks Analysis and Mining (IEEE) 2010, p. 176.

8. Liu F.-C., Choi D., Xie L. and Roeder K., Proc. Natl. Acad. Sci. U.S.A., 115 (2018) 927.

9. Pons P. and Latapy M., Computing Communities in Large Networks Using Random Walks, in Computer and Information Sciences - ISCIS 2005, Lecture Notes in Computer Sciences, Vol. 3733 (Springer, Berlin, Heidelberg) 2005, pp. 284–293.

10. Zhang X.-S. et al., EPL, 87 (2009) 38002.

11. Chakrabarti D., Kumar R. and Tomkins A., Evolutionary clustering, in Proceedings of the 12t IGKDD International Conference on Knowledge Discovery and Data Mining (ACM) 2006, p. 554.

12. Huang J., Nie F.-P. and Huang H., Spectral Rotation versus K-Means in Spectral Clustering, in Proceedings of the 27th AAAI Conference on Artificial Intelligence (AAAI) 2013.

13. MacKay D. J., Information Theory, Inference and Learning Algorithms (Cambridge University Press) 2003.