Identification of Influential Node in an Attributed Citation Network

Project report submitted to the Amrita Vishwa Vidyapeetham in partial fulfilment of the requirement for the Degree of

BACHELOR of TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

Submitted by

Medicherla V N S Abhishek Bharadwaj
AM.EN.U4CSE19135
Jakkinapalli Sampat Srivatsav AM.EN.U4CSE19125
Vukka Rithvik AM.EN.U4CSE19170

AMRITA SCHOOL OF COMPUTING

AMRITA VISHWA VIDYAPEETHAM
(Estd. U/S 3 of the UGC Act 1956)
AMRITAPURI CAMPUS

KOLLAM -690525

MARCH 2023

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

AMRITA VISHWA VIDYAPEETHAM
(Estd. U/S 3 of the UGC Act 1956)

Amritapuri Campus

Kollam -690525

BONAFIDE CERTIFICATE

This is to certify that the project report entitled **"Identification of Influential Node in an Attributed Citation Network"** submitted by **Medicherla V N S Abhishek Bharadwaj(AM.EN.U4CSE19135), Jakkinapalli Sampat Srivatsav(AM.EN.U4CSE19125) and Vukka Rithvik(AM.EN.U4CSE19170))**, in partial fulfillment of the requirements for the award of Degree of Bachelor of Technology in Computer Science and Engineering from Amrita Vishwa Vidyapeetham, is a bonafide record of the work carried out by them under my guidance and supervision at Amrita School of Computing, Amritapuri during Semester 8 of the academic year 2022-2023.

Your Guides Name
Deepthi L R

Coordinator name
Simi S

Dr. Prema Nedungadi
Chairperson
Dept. of Computer Science & Engineering

Reviewer

Place : Amritapuri
Date : 17. 03. 2023

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

AMRITA VISHWA VIDYAPEETHAM
(Estd. U/S 3 of the UGC Act 1956)

Amritapuri Campus

Kollam -690525

DECLARATION

We, **Medicherla V N S Abhishek Bharadwaj(AM.EN.U4CSE19135), Jakkinapalli Sampat Srivatsav(AM.EN.U4CSE19125) and Vukka Rithvik(AM.EN.U4CSE19170)** hereby declare that this project entitled **"Identification of Influential Node in an Attributed Citation Network"** is a record of the original work done by us under the guidance of **Deepthi L R**, Dept. of Computer Science and Engineering, Amrita Vishwa Vidyapeetham, that this work has not formed the basis for any degree/diploma/associationship/fellowship or similar awards to any candidate in any university to the best of our knowledge.

Place : Amritapuri

Date : 17. 03. 2023

M. Abhishek Bharadwaj
J. Sampat Rithvik
Signature of the student

Signature of the Project Guide

Acknowledgements

We would like to extend our sincere gratitude to everyone who helped bring this project to a successful conclusion. First and foremost, we would like to express our profound gratitude to our project mentor, whose direction, assistance, and useful input throughout the project played a crucial role in guiding our work and assisting us in achieving our goals. We appreciate her knowledge, tolerance, and perseverance because these helped us work through the project's difficulties and provide a high-caliber result.

We also want to express our gratitude to the coordinators and academic staff at our university for their assistance and providing of the facilities and tools required to carry out this project. We were inspired to work harder and accomplish our objectives by their support and belief in our skills.We are appreciative of their support and trust, which enabled us to successfully manage the project's difficulties.

We sincerely appreciate our family and friends' unflagging support and encouragement during this undertaking. Their confidence in us, support, and compassion were crucial in keeping us inspired and on task. We appreciate their support and acknowledge that we could not have achieved this goal without their love, tolerance, and understanding. We would like to express our sincere gratitude to everyone who helped this project be completed successfully. We anticipate that our work will aid in other people's endeavors and further the field's study.

Abstract

Researchers have developed various metrics for identifying influential nodes in complex networks, including degree centrality, betweenness centrality, and eigenvector centrality. Degree centrality measures the number of direct connections a node has, while betweenness centrality measures the number of times a node lies on the shortest path between two other nodes. Eigenvector centrality takes into account not only a node's direct connections but also the importance of its neighbors.

Complex systems known as multi-attribute networks are used to record connections between entities based on a variety of attributes. These characteristics can be anything from age to gender to job to hobbies to location. Multi-attribute networks are prevalent across a wide range of fields, such as social, biological, transit, and information networks. An busy field of study, multi-attribute network analysis has a variety of uses, including community detection, link prediction, and anomaly detection. In this summary, we give a general outline of multi-attribute networks, including how they are represented, what they are like, and how to analyze them. Additionally, we go over some of the possibilities and difficulties that come with analyzing multi-attribute networks, including data quality, network growth, and interpretability.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Citation Network

1.1.1 Introduction to Citation Network

- Academic research is a key engine for advancement in science and technology. Researchers require effective and efficient tools to navigate the large and diverse terrain of knowledge as the number of papers published and the complexity of scientific disciplines rise.

- The citation network, which depicts the links between academic articles

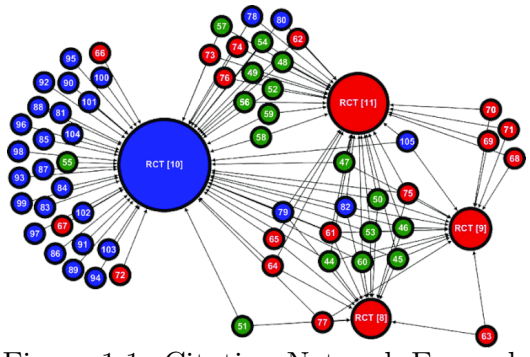

Figure 1.1: Citation Network Example

through their citation relationships, is a crucial tool for this purpose.

- Researchers can investigate the connections between various scientific domains, follow the development of research topics, and spot important contributions and trends using citation networks, which are a great informational resource.

1.1.2 Influential Node in a Citation Network

- A citation network's nodes are not all created equal in terms of significance or influence.
- According to criteria like centrality, degree, or betweenness, some papers, authors, or journals may have an outsized influence on the network structure.
- Finding these key nodes can help with strategic choices about research funding, partnerships, and dissemination while also shedding light on the dynamics and growth of the network.
- With the help of network analysis, machine learning, and data visualization approaches, we hope to create a system for determining influential nodes in a citation network.

1.1.3 Attributes in a Citation Network

- The citation network contains the authors' details and the papers they published. So there are various criteria to find an influential author/paper.
- For instance, we can go for the number of citations for the papers written by the author or the number of co-authors, or the type of publication venue like [conference or journal].

- Our dataset contains a total of 1712433 authors and each author has attributes like H-index,p-index, PC, and CN where each of these depicts the number of papers that were published and the total number of citations of the paper written by the author.
- So initially we can keep some threshold factors to specify or identify the influenced author.

1.1.4 Significance of Influential Node in a Citation Network

- To evaluate the significance of the nodes in the network, we will examine several centrality metrics and graph algorithms.
- We will also train predictive models to categorise nodes as influential or non-influential based on their structural characteristics and information. Also, we'll study and analyse the data using interactive visualisations, and we'll test our methodology on actual citation networks from several scientific fields.
- By attaining this objective, we seek to contribute to the creation of more useful and effective tools for examining and comprehending the intricate and ever-changing world of scholarly research.
- Researchers, publishers, funding organisations, and policy makers are just a few of the stakeholders in the academic ecosystem that our findings may have an impact on.

1.2 Data Acquisition

- The datasets used are AMiner-Author, AMiner-paper, and AMiner-Coauthor.
- The datasets contained details of authors and papers.
- The datasets were converted into a graph representation using Networkx and sage-graph.
- A monolayer network was constructed initially, and later the monolayers were combined to achieve a multilayer citation network.
- The edge weight in the CoAuthor's dataset represents the paper count, i.e., the number of papers shared by authors.
- An attributed graph was constructed based on the author's dataset.
- The main attributes considered for the attributed graph were hi, pi, and upi.
- The author's index was denoted by a unique identifier, such as '522324'.
- The attributes hi, upi, pi were assigned values for each author. The example representation given was:

```
d = {
     '522324': {'#hi': 1, '#pi': 1.3333, '#upi': 0.8222},
     '1355779': {'#hi': 1, '#pi': 0.4000, '#upi': 0.0800}
    }
```

  where hi (h-index) is a metric used to measure the productivity and impact of a researcher's publications.

Chapter 2

Problem Definition

- The difficulty in determining influential nodes in a citation network results from the academic research landscape's increasing complexity.
- A key tool for this is citation networks, which show the connections between academic articles through their citation relationships.
- To make informed choices about research funding, collaborations, and dissemination, it is crucial to recognize the major nodes in a citation network because not all of them are equally important or influential.
- The main objective is to study the network structure and pinpoint the important nodes by combining network analysis, machine learning, and data visualization approaches
- Many reasons make it difficult to pinpoint influential nodes in a citation network.
- First, it may be difficult to evaluate and visualize the network due to its size and complexity, which may include numerous aspects of relationships between nodes.

- Second, different research questions or domains may define node importance differently, necessitating flexible and adaptable methodologies.
- Lastly, the quality and completeness of the data as well as the underlying assumptions and biases in the analysis may affect how accurate and reliable the methods for finding influential nodes are.
- Due to the intricacy of the academic research environment, it is challenging to identify important networks. The number of papers, authors, and links in a given field increases exponentially as research expands and diversifies, making it challenging to determine which nodes (i.e., papers, authors) are genuinely important.
- The network's characteristics, such as its H-index, p-index, and unequal P-index, can also bring new levels of intricacy to the study. Because of this intricacy, nodes that seem important may not actually be so, and vice versa, in a network. As a result, finding the most significant nodes in a citation network necessitates a thorough evaluation of the structure and characteristics of the network as well as the researcher's experience and judgment.

Chapter 3

Related Work

It is possible to identify influential authors in networks by various means, such as analyzing their characteristics or the number of co-authors. However, it is necessary to select or develop algorithms that use the author's attributes to accurately identify influential nodes. Some papers have dealt with this subject, some of which are below.

3.1 Influential Nodes Identification in Complex Networks via Information Entropy

- In this paper [1], the authors introduced the concept of multilayer networks and information entropy.
- They defined structural holes as gaps between a collection of indirectly connected nodes, with intermediaries acting as mediators for the exchange of information.
- The authors proposed a simple algorithm based on degree and structural hole count to evaluate node relevance in preserving network con-

nectivity.

3.2 Vital nodes identification in complex networks

- Identifying significant nodes in complex networks is crucial to optimize the structure of the network.
- In this paper [2], the authors propose an improved approach of identifying influential nodes that uses a hybrid mechanism of information entropy and weighted edge to increase identification accuracy.
- The proposed method was superior in performance to other methods, as demonstrated by theoretical analysis and experimental results on real data sets.
- This approach can be used with large networks because of its low computing complexity and can be applied in various fields such as rumor control and advertising targeting.

3.3 A Novel Centrality of Influential Nodes Identification in Complex Networks

- In this paper [3], the authors presented a novel centrality, called DCC, to discover prominent nodes by taking into account degree, clustering coefficient, and neighbors. Entropy technology is used to calculate the clustering coefficient and degree weights.

3.4 Tensor-based Study of Eigenvectors in Multi-layer Network

- Centrality is a key metric for understanding and managing the process of spreading in complex networks.
- In this study [9], the authors used a tensor-based framework to study the multi-centrality of the eigenvector.
- This approach provides a methodical way of explaining how multi-centralization spreads over several layers.
- Two interesting scenarios are provided to show how to describe multi-layered influence. The analysis of various empirical multi-layer networks using this method shows its effectiveness

3.5 Improved Influential Node Identification

- Clustering a graph, or grouping its nodes, is crucial and is most commonly used to identify communities in social networks.
- With the significant exception of edge weights, graph clustering, and community detection has traditionally focused on graphs without attributes.
- Real social systems, on the other hand, are frequently characterized using node attributes, which reflect the characteristics of the players, and edge attributes, which represent various types of relationships among them.

- These models, however, only give a partial depiction of real social systems. These models are referred to as attributed graphs. As a result, node and edge attributes have lately been included in the scope of existing graph clustering techniques.

- This article provides a review of the literature on the subject, arranging and presenting the latest research finding uniformly, describing the primary clustering techniques now in use, and highlighting their conceptual distinctions.

- We also discuss the crucial subject of clustering evaluation and point out any unanswered issues at the moment

3.6 Node Discovery Based on Local Community

- Recognizing local community structure in networks is crucial, and several recent studies have addressed this topic.

- However, the majority of current methods demand total knowledge of the graph's structure, which makes the unworkable for some networks.

- This research suggests a novel critical node identifying an approach for anchor complex networks that is based on local community aggregation and recognition.

- The suggested method efficiently mines and identifies significant nodes based on a specific local community and enhances the quality of community detection in comparison to other cutting-edge algorithms.

- The proposed method has been demonstrated through computer-generated networks and real-network experiments.

3.7 Clustering attributed graphs models, measures and methods

- The process of grouping graphs or clustering nodes is vital to the identification of social network communities.
- With the exception of edgeweight, the classification and community detection of graphs are traditionally based on graphs without attributes.
- On the other hand, real social systems are often characterized by node attributes that reflect player characteristics and edge attributes that represent a variety of relationships between them.
- , However, these models only partially represent the real social system.These models are called attributed graphs. Consequently, the attributes of nodes and edges have recently been included in the scope of existing graph clustering techniques.
- This paper reviews the literature on this subject uniformly presents the latest research findings, describes the main clustering techniques currently used,and emphasizes their conceptual differences.
- They also discussed the critical issue of clustering evaluation and highlighted any questions that are currently unresolved

3.8 Community Detection in Attributed Network

- They divided the present approaches to the attributed clustering problem into three major categories.
- Using a set of synthetic and actual data, they compared a set of attributed network community discovery algorithms.
- According to experimental findings, algorithms that include both types of information can successfully cluster vertices into informative clusters.
- This paper is structured as follows to investigate the above task. The attributed network clustering problem is first introduced. They provided a taxonomy of the state-of-the-art techniques currently being used to address this issue.
- The experimental evaluation of a network community detection approach that is currently in use is then provided.

Chapter 4

Requirements

The design of this project contains both hardware and software. The specifications that we used are listed below.

4.1 Hardware

Machine type	e2-custom-10-40960
CPU platform	Intel Broadwell
Architecture	x86/64
vCPUs to core ratio	2 vCPU per core
RAM	96 GB
Storage	64 GB
Operating System	Ubuntu 22.04 LTS

Table 4.1: Hardware Requirements

4.2 Software

Languages	Python 3.10
Modules	NetworkX, TQDM, dash, nltk, scikit learn, pandas, numpy, matplotlib, MPL-toolkit
Tools	SageMath, plotly
Platforms	Azure, AWS, Google Cloud Platform

Table 4.2: Software Requirements

Chapter 5

Proposed System

we've seen the difficulty in determining influential nodes due to the complexity of the academic research landscape. As research continues to expand and diversify, the number of papers, authors, and citations in a given field grows at an exponential rate, making it increasingly difficult to identify which nodes (i.e., papers, authors) are truly influential. Furthermore, the attributes of the network (e.g., H-index, p-index, unequal P-index) can add additional layers of complexity to the analysis. This complexity can result in a network where nodes that appear influential may not necessarily be so, and vice versa. Thus, identifying truly influential nodes in a citation network requires careful consideration of both the network's structure and its attributes, as well as the researcher's expertise and judgment. Let's see some approaches for the identification of the critical nodes

5.1 Centralities

Centrality measures are used to identify the most important nodes or vertices in a network. Here are some of the most commonly used centrality measures.

5.1.1 Degree Centrality

A node's degree of centrality is determined by counting the connections that node has, which is the most basic way to evaluate centrality. High-degree centrality nodes have a lot of connections to other nodes in the network.

$$C_D(v) = k(v) where\ k(v)\ is\ the\ degree\ of\ node\ v.$$

5.1.2 Betweenness Centrality

The notion that nodes that are located on numerous shortest routes between other nodes are more significant is the foundation of betweenness centrality. It is determined by counting how many times a node appears on the network's shortest route between other nodes.

$$C_B(v) = \sum_{s \neq v \neq t} \frac{\sigma(s,t|v)}{\sigma(s,t)}$$

where $\sigma(s,t)$ is the number of shortest paths from node s to node t, and $\sigma(s,t|v)$ is the number of shortest paths from node s to node t that pass through node v.

5.1.3 Eigenvector Centrality

This measure of a node's importance takes into consideration the caliber of its connections. Based on the notion that a node is essential if it is connected to other important nodes, this calculation is made. Nodes connected to other nodes with high centrality receive greater scores from the measure.

$$C_E(v) = \frac{1}{\lambda} \sum_u A(v,u) C_E(u)$$

where λ is a constant, $A(v, u)$ is the adjacency matrix element for node v and u, and $C_E(u)$ is the eigenvector centrality of node u.

5.1.4 Closeness Centrality

The average distance between a node and every other node in the network is used to calculate closeness centrality. The network's nodes that can reach other nodes more rapidly than others have high closeness centrality.

$$C_C(v) = \frac{n-1}{\sum_{u \neq v} d(u, v)}$$

where n is the total number of nodes in the network, and $d(u, v)$ is the shortest path distance between nodes u and v.

5.2 Text Similarities

5.2.1 Cosine Similarity

- A measure of similarity between two non-zero vectors in an inner product space is called cosine similarity.
- It is often used in natural language processing (NLP) to determine the similarity between two text documents. Cosine similarity is calculated as follows:

$$\text{cosine similarity} = S_C(A, B) := \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}},$$

where "." signifies the dot product, "|| ||" denotes the Euclidean norm,

and a and b are the two vectors being compared.

- The vectors a and b in the context of text similarities represent the phrase frequency of terms in the two documents under comparison.
- The two vectors' cosine similarity measures between -1 and 1, with 1 denoting total similarity and -1 denoting complete dissimilarity. **Here are the steps involved in using cosine similarity for text-similarities:**

5.2.2 Vectorization

- Create a vector representation of each document's phrase frequency.
- Cosine similarity calculation: Using the aforementioned procedure, determine the cosine similarity between the two documents' vector representations.
- Interpretation: Cosine similarity scores can be interpreted to assess how similar two documents are.
- Several NLP tasks, including document grouping, information retrieval, and recommendation systems, make extensive use of cosine similarity.
- It is an effective tool for determining text similarity, and it can be combined with other methods to increase measurement accuracy.

5.2.3 Tokenization

- The process of turning text documents into a list of tokens, or individual words, is known as tokenization.
- To create a vocabulary of all the distinct words in the corpus, the text is divided into individual words or terms in this process.

- Tokens can be produced in a variety of ways, such by dividing the text into tokens based on punctuation and white space or by utilizing NLTK libraries for natural language processing (Natural Language Toolkit).

5.2.4 Text preprocessing

- Prior to processing, the text data must be cleaned.
- Lemmatization, stemming, and the elimination of stop words are some of the steps in this process.
- Stop words, such as "the," "and," "in," etc., are frequently used terms in a language that are not helpful for text analysis. we can see these stopwords in the affiliations
- Lemmatization and stemming are methods for getting words back to their base form so that different spellings of the same word can be used interchangeably.

5.2.5 Term frequency calculation

- It represents the frequency with which a term occurs in a document.
- We count the instances of each token in the document in this stage.
- A matrix of term frequency is created for each text in the corpus using this data.

5.2.6 Vectorization

- It involves turning each document's phrase frequency into a vector representation.

- Several techniques, such as the Bag of Words (BoW) or Term Frequency-Inverse Document Frequency, can be used to create the vectors (TF-IDF).
- Each document is represented by a vector representing the frequency of each word in the lexicon in the BoW model.
- Each document is represented by the TF-IDF model as a vector of the product of term frequency and inverse document frequency, where the latter is used to reduce the weight of terms that often occur in the corpus.
- As a result, cosine similarity is a strong instrument for assessing text similarity and is frequently utilized in a variety of NLP tasks.
- Cosine similarity can assist in precisely determining the similarity between text documents by breaking down the text documents into individual tokens, preprocessing the text, and transforming the term frequency of each document into a vector representation.

5.3 Clustering

- Before moving to clustering, we have considered attributes such as hi,pi,upi and affiliations as key factors for finding the influential author.
- For these we formulated for combining the attributes hi,pi,upi and the formula goes as

```
(5 * e^(ln(|d[i]["#hi"]|)) * (cos(arg(d[i]["#hi"]))
```

```
    + i*sin(arg(d[i]["#hi"])))
    + e^(ln(|d[i]["#upi"]|)) * (cos(arg(d[i]["#upi"]))
    + i*sin(arg(d[i]["#upi"])))
    + e^(ln(|d[i]["#pi"]|)) * (cos(arg(d[i]["#pi"]))
  + i*sin(arg(d[i]["#pi"])))) / 3
```

- this formula indeed can be solved like this : $e^{\ln(|a|)} = |a|$, because $e^{\ln(x)} = x$ for any positive real number x.
- Similarly, $e^{\ln(|b|)} = |b|$ and $e^{\ln(|c|)} = |c|$.
- The term $\cos(\arg(a))+i\sin(\arg(a))$ represents the complex number a in polar form, where $\arg(a)$ is the argument (or angle) of a and $\cos(\arg(a))$ and $\sin(\arg(a))$ are the real and imaginary parts of a, respectively. We can write this as $a = |a| \cdot e^{i\arg(a)}$.
- Similarly, the terms $\cos(\arg(b))+i\sin(\arg(b))$ and $\cos(\arg(c))+i\sin(\arg(c))$ represent the complex numbers b and c, respectively, in polar form.
- We can combine the terms by using the distributive property of multiplication:

  $5(e^{\ln(|a|)})\cdot(\cos(\arg(a))+i\sin(\arg(a))) = 5|a|(\cos(\arg(a))+i\sin(\arg(a))) = 5a$
- Similarly, $e^{\ln(|b|)}\cdot(\cos(\arg(b))+i\sin(\arg(b))) = b$, and $e^{\ln(|c|)}\cdot(\cos(\arg(c))+i\sin(\arg(c))) = c$.
- Adding the terms and dividing by 3, we get:

$$\frac{5a+b+c}{3}$$

so now we can use several clustering algorithms for grouping these attributes into different communities

5.3.1 K-Means-Clustering

- K-means clustering is a machine-learning technique used for partitioning a given dataset into K clusters or subgroups.
- The goal is to divide the data-set in such a way that the objects within each cluster are as similar as possible to each other, while objects in different clusters are as dissimilar as possible.
- The output of the algorithm is a set of K clusters, each with its own centeroid and the data points assigned to it. we used Elbow method for determining the value of the K

5.3.2 DBSCAN

- DBSCAN works by partitioning the data space into regions of high density, based on a distance metric between data points, and then identifying clusters as connected components within these dense regions.
- It requires two main parameters: a distance threshold (epsilon) and a minimum number of points required to form a dense region (minPts).
- DBSCAN partitions data space in to high density regions based on distance meters between data points and then determines clusters that are connected components within these dense regions.
- It requires two main parameters: a distance threshold (epsilon) and a minimum number of points(minPts) for forming a dense region. Modu-

larity Score : Modularity scores are the quality of community structures in networks and graphs.

- The modularity score measures the extent to which the number of edges in a community is higher than expected by chance, given the distribution ofdegree in the community. A higher modular score indicates a better community structure.

$$Q = \frac{1}{2m} \sum_{i,j} \left(A_{ij} - \gamma \frac{k_i k_j}{2m} \right) \delta(c_i, c_j)$$

where:

Q is the modularity score

m is the total number of edges in the network

A_{ij} is the weight of the edge between nodes i and j

k_i and k_j are the degrees of nodes i and j, respectively

c_i and c_j are the communities to which nodes i and j belong, respectively

$\delta(c_i, c_j)$ is the Kronecker delta function, which is 1 if i and j belong to the same community, and 0 otherwise

γ is the resolution parameter, which controls the size of the communities

5.3.3 Hierarchical clustering

- A clustering method that builds a hierarchy of groups is called hierarchical clustering.

- It is a bottom-up method where each data point begins as its own cluster and then clusters are gradually combined based on a similarity criterion until a single cluster comprising all the data points is created.

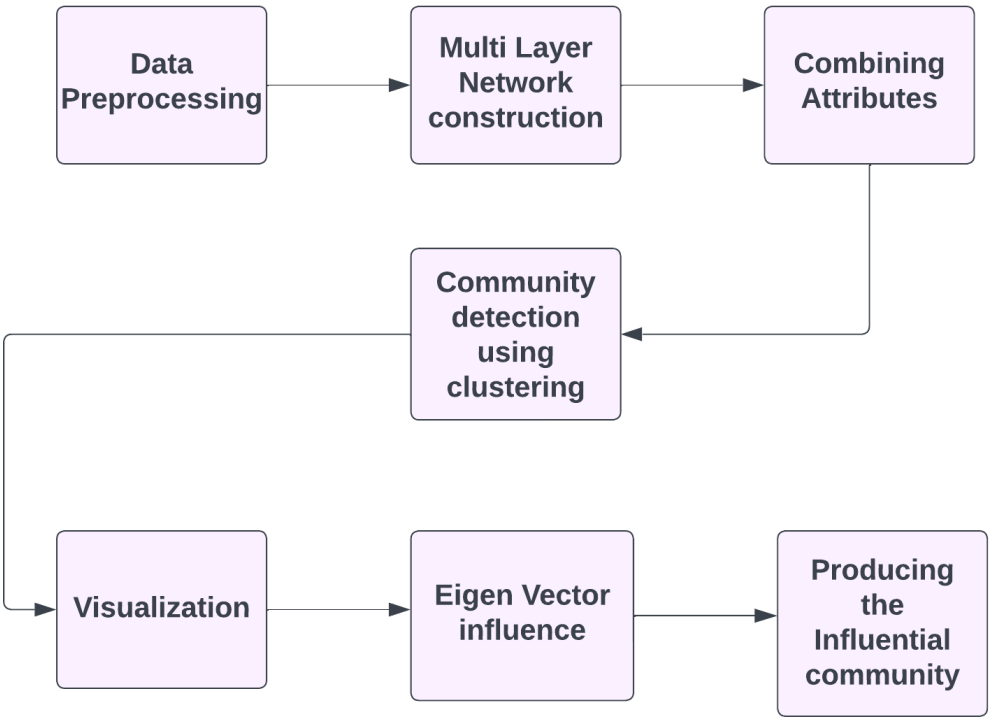

Figure 5.1: Block Diagram

- An illustration of the resulting hierarchy that illustrates the connections between clusters at various stages of the hierarchy is called a Dendrogram.
- Each data point is originally taken into account as a distinct cluster in the bottom-up method known as agglomerative clustering, after which pairs of clusters are combined together based on a similarity measure until all of the data points are members of the same cluster.
- Divisive clustering is a top-down strategy in which all the data points are originally regarded as belonging to the same cluster before being successively divided into separate groups according to a dissimilarity criterion.

Chapter 6

Result and Analysis

Firstly we successfully constructed a multilayer citation network using these single layers. In the attributed graph we no longer require the authors who have the H-index as 0 so we removed those entries in the dataset.

6.1 Testing

- In order to evaluate the effectiveness of community and clustering techniques for identifying prominent nodes in an ascribed citation network, data preparation is a crucial step. To make the data uniform and suitable for analysis, it must first be cleaned and standardized. Remove duplicates, normalize attribute values, and screen out nodes with low degree or low reference count are a few typical preprocessing methods. In order to simplify analysis and decrease the number of variables, it may also be essential to employ dimensionality reduction methods like principal component analysis (PCA) . After preprocessing the data, it can be examined using community and clustering techniques to determine the most important networks. grouping strategies.

- After preprocessing the data, it can be examined using community and clustering techniques to determine the most important networks. For community identification and for attribute-based clustering, they might use k-means or hierarchical clustering. Using centrality metrics like degree centrality or betweenness centrality, the found communities and clusters can be further analyzed to find important individuals. Using measures like modularity, recall, or F1 score, the found influential nodes can then be verified against known influential nodes in the citation network. In general, accurate data preparation is essential for identifying important nodes using the community and clustering technique and accurately analyzing the citation network.

- Additionally, choosing appropriate evaluation metrics is important to assess the quality of the identified communities and clusters, as well as the accuracy of the identified influential nodes.

- Finally, because bigger datasets may necessitate more numerically demanding techniques, it is crucial to take the algorithms' computational effectiveness into account. Overall, when finding important nodes in an ascribed citation network using community and clustering techniques, careful consideration of algorithm selection and assessment measures can increase the precision and quality of the outcomes.

6.2 Results

- Network Visualization: Network research tools like MPL tool kit or Gephi can be used to visualize the reference network. Finding important nodes, clusters, and communities that are interconnected and powerful can be done with the aid of network visualization.

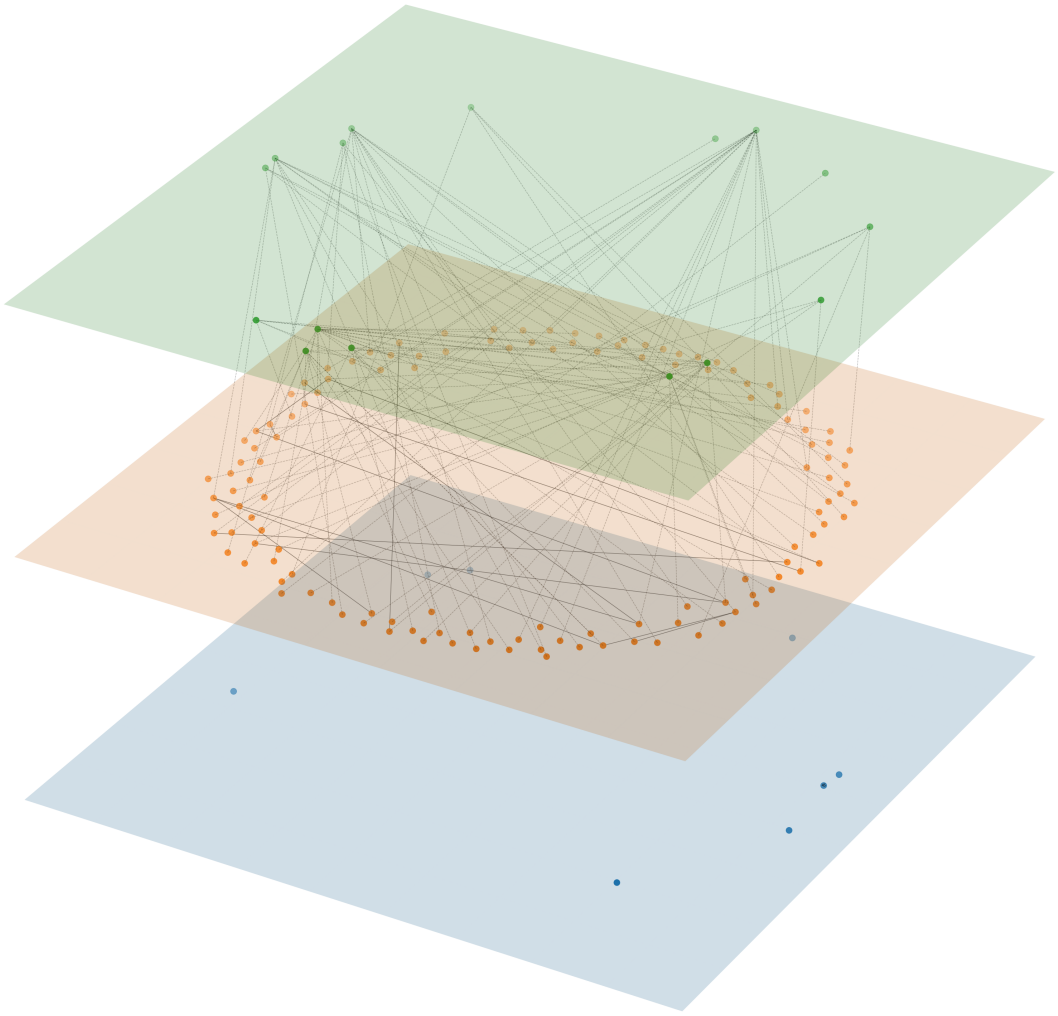

Figure 6.1: Multi-Layer Clustering

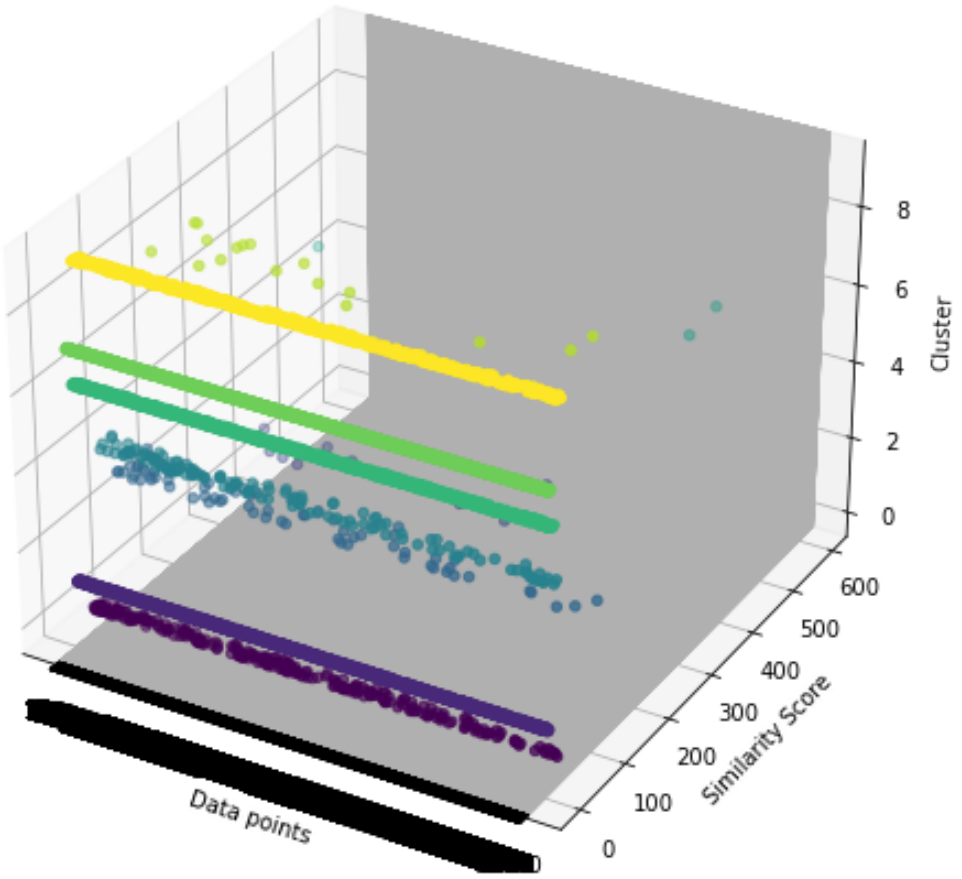

Figure 6.2: K-Means CLustering

- Clustering Methods: we've grouped comparable nodes together based on their characteristics, by using clustering techniques like k-means or DBSCAN. (e.g., attribute-sum, similarity score). This can assist you in finding groups of nodes with comparable characteristics and identifying the nodes that are most important within these groups.

- Evaluation metrics: Present the evaluation metrics used to assess the quality of the analysis, including any modularity, silhouette coefficient, or Rand index values obtained. we've got a modularity score of around 96

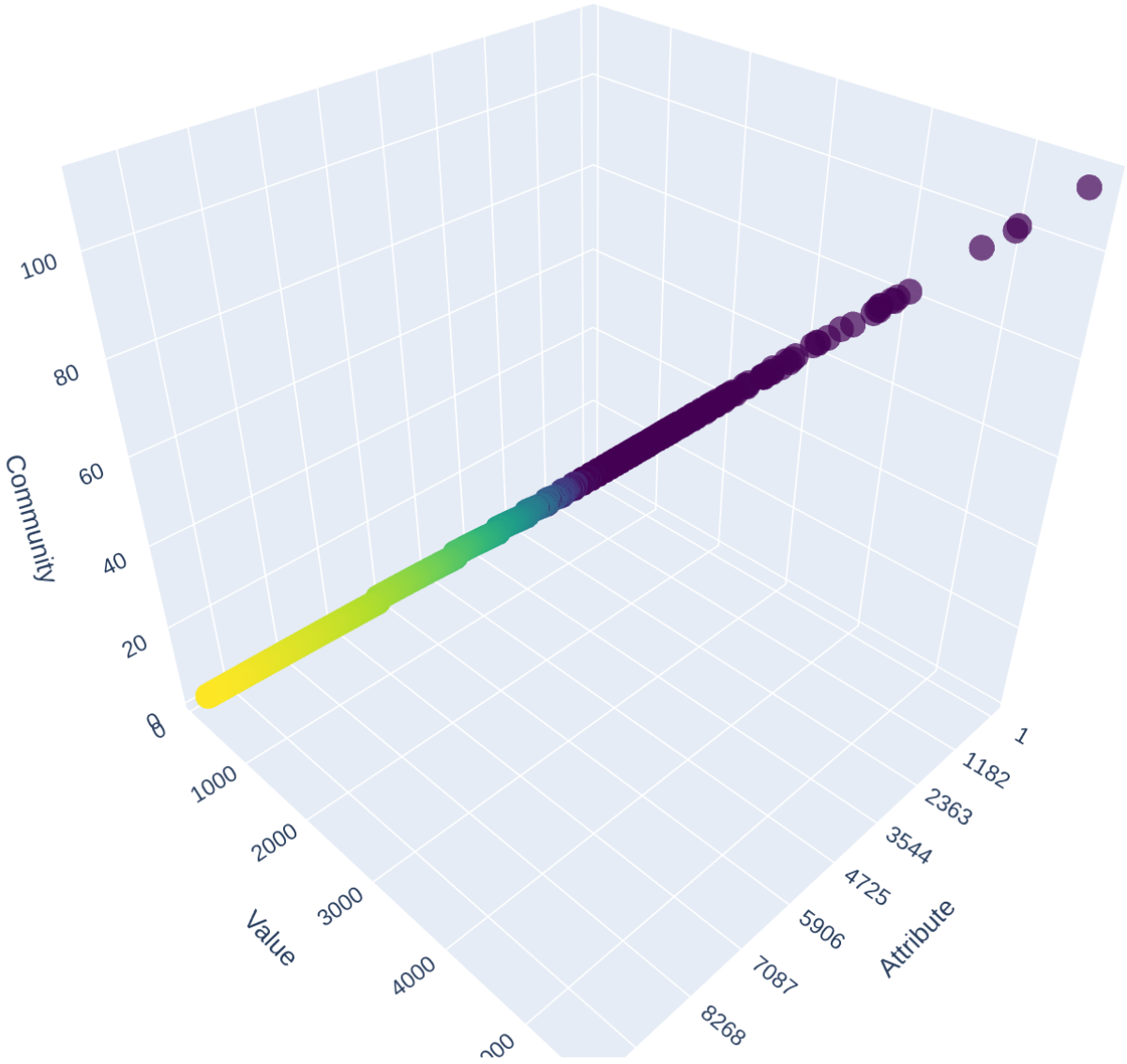

Figure 6.3: DBScan Output

Chapter 7

Conclusion

- In conclusion, Using community and clustering techniques, it is possible to locate significant nodes in an attributed citation network, which can reveal important details about the network's behavior and structure. Researchers can better understand the dissemination of ideas and the emergence of research partnerships within the network by finding highly linked communities or clusters of nodes with similar characteristics. The most significant papers, authors, or organizations within the network can also be recognized by the discovered influential nodes, which can be helpful for future study or decision-making. However, when dealing with credited reference networks, it is crucial to take security and ethical factors into account, such as safeguarding the anonymity of people and organizations engaged in the research and making sure the research is carried out in line with accepted ethical standards.

- Additionally, the discovery of significant nodes within the network can offer useful information for academics and professionals working in a variety of disciplines. Researchers can get an understanding of the present status of the field's study as well as possible directions for future

research by finding the most significant papers, authors, or institutions within the network. This knowledge can also be helpful for making decisions, such as finding prospective partners or funding options.

- Overall, identifying influential nodes in an attributed citation network using community and clustering methods can be a powerful tool for understanding the structure and dynamics of citation networks and can provide valuable insights for researchers and practitioners in various fields.

References

[1] C. Guo, L. Yang, X. Chen, D. Chen, H. Gao, and J. Ma, "Influential Nodes Identification in Complex Networks via Information Entropy" , https://doi.org/10.3390/e22020242

[2] Linyuan Lü, Duanbing Chen, Xiao-Long Ren, Qian-Ming Zhang, Tao Zhou, "Vital nodes identification in complex networks" , https://arxiv.org/abs/1607.01134

[3] Yang, Yuanzhi et al. "A Novel Centrality of Influential Nodes Identification in Complex Networks.", https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9045973&tag=1

[4] Batagelj, Vladimir "Efficient Algorithms for Citation Network Analysis", https://arxiv.org/abs/cs/0309023

[5] Guo, C.; Yang, L.; Chen, X.; Chen, D.; Gao, H.; Ma, J. "Influential Nodes Identification in Complex Networks via Information Entropy", https://www.mdpi.com/1099-4300/22/2/242

[6] Francisco Pedroche, Esther García, Miguel Romance, Regino Criado, "Sharp estimates for the personalized Multiplex PageRank, Journal of Computational and Applied Mathematics", https://www.sciencedirect.com/science/article/pii/S0377042717300717

[7] Mincheng Wu, Shibo He, Yongtao Zhang, Jiming Chen, Youxian Sun, Yang-Yu Liu, Junshan Zhang, H. Vincent Poor, "A Tensor-Based Framework for Studying Eigenvector Multicentrality in Multilayer Networks", https://arxiv.org/abs/1708.07763

[8] Christoph Rahmede, Jacopo Iacovacci, Alex Arenas, Ginestra Bianconi, "Centralities of Nodes and Influences of Layers in Large Multiplex Networks", https://arxiv.org/abs/1703.05833

[9] Sune Lehmann Jørgense, Complex Spreading Phenomena in Social Systems: Influence and Contagion in Real-World Social Networks, https://orbit.dtu.dk/en/publications/complex-spreading-phenomena-in-social-systems-influence-and-conta

[10] L. Wan, M. Zhang, X. Li, L. Sun, X. Wang and K. Liu, "Identification of Important Nodes in Multilayer Heterogeneous Networks Incorporating Multirelational Information,", https://ieeexplore.ieee.org/document/9756648

Appendix A

Source code

A.0.1 Data Parsing

```
import pandas as pd
from tqdm import *
import os
import glob
import csv
from xlsxwriter.workbook import Workbook


get_ipython().system("python3 -m  pip install pandas")
ls
authordata = "output1.txt"
file1 = open(authordata, "r")
lines = file1.readlines()
len(lines)
authId = []
authname = []
```

```
pubpaper = []
affilications = []
citation_count = []
h_index = []
interests = []
pindequal = []
pindexinequ = []
for i in range(len(lines)):
    if lines[i][0] == "#":
        if lines[i][1] == "i":
            authId.append(lines[i][7:-1])
        if lines[i][1] == "n":
            authname.append(lines[i][3:-1])
        if lines[i][1:3] == "pc":
            pubpaper.append(lines[i][4:-1])
        if lines[i][1:3] == "hi":
            h_index.append(lines[i][4:-1])
        if lines[i][1] == "a":
            affilications.append(lines[i][3:-1])
        if lines[i][1:3] == "pi":
            pindequal.append(lines[i][4:-1])
        if lines[i][1:4] == "upi":
            pindexinequ.append(lines[i][5:-1])
        if lines[i][1] == "t":
            interests.append(lines[i][3:-1])
        if lines[i][1:3] == "cn":
            citation_count.append(lines[i][4:-1])
```

```
print(len(authId), authId[0], authId[-1])
print(len(authname), authname[0], authname[-1])
print(len(pubpaper))
print(len(affilications))
print(len(citation_count))
print(len(h_index))
print(len(interests))
author_df = pd.DataFrame(
    list(
        zip(
            authId,
            authname,
            pubpaper,
            affilications,
            citation_count,
            h_index,
            interests,
        )
    ),
    columns=[
        "AuthorId",
        "Author Name",
        "Paper Publications",
        "Afflications",
        "Citation Count",
        "H Index",
        "Paper of interests",
```

```
    ],
)
author_df.to_csv("author.csv")
author_df.drop_duplicates(inplace=True)
author_df
papertxtfile = open("paper.txt", "r")
paper_lines = papertxtfile.readlines()
papertxtfile1 = open("AMiner-Paper.txt", "r")
paper_lines1 = papertxtfile1.readlines()
paper_lines[:2]
paper_lines1[2]
index = []
papertitle = []
authors = []
affiliations = []
year = []
publicationvenue = []
idref = []
abstract = []
ind_lst = list()
for i in trange(1000000):
    if paper_lines[i] == "\n":
        if paper_lines[i - 1][1] != "!" and paper_lines[i - 1][1] != "%":
            idref.append(0)
            abstract.append("null")
        elif paper_lines[i - 1][1] != "!" and paper_lines[i - 1][1] == "%":
            idref.append(ind_lst)
```

```
            ind_lst = []
            abstract.append("null")
        elif paper_lines[i - 1][1] == "!" and paper_lines[i - 2][1] != "%":
            idref.append(0)
        else:
            pass
    elif paper_lines[i][0] == "#":
        if paper_lines[i][1] == "i":
            index.append(paper_lines[i][7:-1])
        if paper_lines[i][1] == "*":
            papertitle.append(paper_lines[i][2:-1])
        if paper_lines[i][1] == "@":
            authors.append(paper_lines[i][2:-1])
        if paper_lines[i][1] == "o":
            affiliations.append(paper_lines[i][2:-1])
        if paper_lines[i][1] == "t":
            year.append(paper_lines[i][2:-1])
        if paper_lines[i][1] == "c":
            publicationvenue.append(paper_lines[i][2:-1])
        if paper_lines[i][1] == "%":
            ind_lst.append(int(paper_lines[i][3:-1]))
        if paper_lines[i][1] == "!" and paper_lines[i - 1][1] == "%":
            idref.append(ind_lst)
            ind_lst = []
            abstract.append(paper_lines[i][2:-1])
        if paper_lines[i][1] == "!" and paper_lines[i - 1][1] != "%":
            abstract.append(paper_lines[i][2:-1])
```

```
print("index", len(index))
print("papertitle", len(papertitle))
print("authors", len(authors))
print("affiliations", len(affiliations))
print("year", len(year))
print("ref", len(idref))
print("abstract", len(abstract))
data1 = pd.DataFrame(
    list(
        zip(
            index,
            papertitle,
            authors,
            affiliations,
            year,
            publicationvenue,
            idref,
            abstract,
        )
    ),
    columns=[
        "Index",
        "Paper Title",
        "Authors",
        "Afflications",
        "Year",
        "Venue",
```

```
        "Idref",
        "abstract",
    ],
)
data1
data1.to_csv("fullpaper.csv")
get_ipython().system("pip install xlsxwriter")
for csvfile in glob.glob(os.path.join(".", "*.csv")):
    workbook = Workbook(csvfile[:-4] + ".xlsx")
    worksheet = workbook.add_worksheet()
    with open(csvfile, "rt", encoding="utf8") as f:
        reader = csv.reader(f)
        for r, row in enumerate(reader):
            for c, col in enumerate(row):
                worksheet.write(r, c, col)
    workbook.close()
coauthortxtfile = "output3.txt"
with open(coauthortxtfile, "r") as file:
    readline = file.read().split("\n")
lines = []
for i in readline:
    a = i.split()
    lines.append(a)
indexidofauthor1 = []
indexidofauthor2 = []
numofcollaborations = []
for i in range(len(lines) - 1):
```

```
    indexidofauthor1.append(lines[i][0][1:-1])
    indexidofauthor2.append(lines[i][1])
    numofcollaborations.append(lines[i][2])
Coauthordf = pd.DataFrame(
    list(zip(indexidofauthor1, indexidofauthor2, numofcollaborations)),
    columns=[
        "Index of Author",
        "Index of Co-author",
        "Number of Collaborations between them",
    ],
)
Coauthordf
Coauthordf.to_csv("coauthor.csv")
```

A.0.2 Multi-Layer Construction

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import networkx as nx
from mpl_toolkits.mplot3d import Axes3D
from mpl_toolkits.mplot3d.art3d import Line3DCollection

paper_data = "fullpaper.csv"
author_data = "author.csv"
coauth_data = "coauthor.csv"
auth_df = pd.read_csv(author_data)
```

```
fullpaper_df = pd.read_csv(paper_data)
fullpaper_df.tail()
auth_df.head()
auth_df.head()
affili = auth_df.iloc[10][4]
print(type(affili))
print(affili)
auth_df["Afflications"] = auth_df["Afflications"].fillna("")
auth_df = auth_df.dropna(subset=["Afflications"])
auth_df["Afflications"].str.contains(affili.lower().strip())
s = 0
auth_id = auth_df[auth_df["Afflications"].str.contains(affili)]
auth_id
print(auth_df.iloc[0][2])
auth_nx = nx.Graph()
for i in range(len(auth_id)):
    auth_nx.add_node(auth_id.iloc[i][1], name=auth_id.iloc[i][2])
auth_nx.nodes()
auth_node_attr = nx.get_node_attributes(auth_nx, "aff")
auth_list = list(auth_node_attr.keys())
a_v_list = list(auth_node_attr.values())
print(auth_list)
print(a_v_list)
user_ip = "Software Engineering"
paper_df = fullpaper_df[fullpaper_df["Paper Title"]
.str.contains(user_ip.lower())]
paper_df.iloc[0][2]
```

```
paper_df.iloc[0][8]
len(paper_df)
paper_df.head(1)
paper_nx = nx.Graph()
for i in range(len(paper_df)):
    if paper_df.iloc[i][3] != " ":
        paper_nx.add_node(
            paper_df.iloc[i][1],
            name=list(paper_df.iloc[i][3].split(";")),
            year=paper_df.iloc[i][5],
            ref=paper_df.iloc[i][7],
        )
paper_nx.nodes()
paper_node_attr = nx.get_node_attributes(paper_nx, "ref")
paper_list = list(paper_node_attr.keys())
p_v_list = list(paper_node_attr.values())
print(paper_list)
print()
print(p_v_list)
len(paper_df)
year_nx = nx.Graph()
for _, row in paper_df.iterrows():
    year = row[5]
    if year.strip():
        year_nx.add_node(year, year=year, name=year)

year_nx.nodes()
```

```
coauth_df = pd.read_csv(coauth_data)
coauth_df
```

```
class LayeredNetworkGraph(object):
    def __init__(self, graphs, node_labels=None,
     layout=nx.spring_layout, ax=None):
        self.graphs = graphs
        self.total_layers = len(graphs)
        self.node_labels = node_labels
        self.layout = layout
        if ax:
            self.ax = ax
        else:
            fig = plt.figure()
            self.ax = fig.add_subplot(111, projection="3d")
        self.get_nodes()
        self.get_edges_within_layers()
        self.get_edges_between_layers()
        self.get_node_positions()
        self.draw()

    def get_nodes(self):
        """Construct an internal representation of nodes with the
        format (node ID, layer)."""
        self.nodes = []
        for z, g in enumerate(self.graphs):
```

```
            self.nodes.extend([(node, z) for node in g.nodes()])

    def get_edges_within_layers(self):
        """Remap edges in the individual layers to the internal
        representations of the node IDs."""
        self.edges_within_layers = []
        p_p_edge = list()
        for z, g in enumerate(self.graphs):
            if z == 1:
                paper_node_attr = nx.get_node_attributes(paper_nx
                , "ref")
                paper_list = list(paper_node_attr.keys())
                p_v_list = list(paper_node_attr.values())
                for indx, x in enumerate(p_v_list):
                    if x != "0":
                        y = x[1:-1].split(",")
                        for ref_p in y:
                            pos = p_v_list.index(x)
                            if int(ref_p) in paper_nx.nodes():
                                p_p_edge.append(((paper_list[pos],
                                 z), (int(ref_p), z)))
            else:
                self.edges_within_layers.extend(
                    [((source, z), (target, z)) for (source,
                     target) in g.edges()]
                )
            self.edges_within_layers.extend(p_p_edge)
```

```
def get_edges_between_layers(self):
    self.edges_between_layers = []
    layered_edges = list()
    p_y_edges = list()
    for z1, g in enumerate(self.graphs[:-1]):
        z2 = z1 + 1
        h = self.graphs[z2]
        shared_nodes = set(g.nodes()) & set(h.nodes())
        if z1 == 0:
            paper_node_attr = nx.get_node_attributes(paper_nx, "name")
            author_node_attr = nx.get_node_attributes(auth_nx, "name")
            key_list = list(author_node_attr.keys())
            val_list = list(author_node_attr.values())
            for i in paper_node_attr.items():
                for x in i[1]:
                    found_ath = x.strip() in val_list
                    if found_ath == True:
                        pos = val_list.index(x.strip())
                        layered_edges.append(((key_list[pos], z1),
                         (i[0], z2)))
            self.edges_between_layers.extend(layered_edges)
        if z1 == 1:
            paper_node_attr = nx.get_node_attributes(paper_nx, "year")
            year_node_attr = nx.get_node_attributes(year_nx, "year")
            year_list = list(year_node_attr.values())
            for sngpaper in paper_node_attr.items():
```

```
                if sngpaper[1] in year_list:
                    pos = year_list.index(sngpaper[1])
                    p_y_edges.append(((sngpaper[0], z1),
                     (year_list[pos], z2)))
            self.edges_between_layers.extend(p_y_edges)

    def get_node_positions(self, *args, **kwargs):
        composition = self.graphs[0]
        for h in self.graphs[1:]:
            composition = nx.compose(composition, h)
        pos = self.layout(composition, *args, **kwargs)
        self.node_positions = dict()
        for z, g in enumerate(self.graphs):
            self.node_positions.update(
                {(node, z): (*pos[node], z)
                for node in g.nodes()}
            )

    def draw_nodes(self, nodes, *args, **kwargs):
        (x, y, z) = zip(*[self.node_positions[node]
        for node in nodes])
        self.ax.scatter(x, y, z, *args, **kwargs)

    def draw_edges(self, edges, *args, **kwargs):
        segments = [
            (self.node_positions[source],
            self.node_positions[target])
```

```
            for (source, target) in edges
        ]
        line_collection = Line3DCollection(segments,
         *args, **kwargs)
        self.ax.add_collection3d(line_collection)

    def get_extent(self, pad=0.1):
        xyz = np.array(list(self.node_positions.values()))
        (xmin, ymin, _) = np.min(xyz, axis=0)
        (xmax, ymax, _) = np.max(xyz, axis=0)
        dx = xmax - xmin
        dy = ymax - ymin
        return ((xmin - pad * dx, xmax + pad * dx),
         (ymin - pad * dy, ymax + pad * dy))

    def draw_plane(self, z, *args, **kwargs):
        ((xmin, xmax), (ymin, ymax)) = self.get_extent(pad=0.1)
        u = np.linspace(xmin, xmax, 10)
        v = np.linspace(ymin, ymax, 10)
        (U, V) = np.meshgrid(u, v)
        W = z * np.ones_like(U)
        self.ax.plot_surface(U, V, W, *args, **kwargs)

    def draw_node_labels(self, node_labels, *args, **kwargs):
        for node, z in self.nodes:
            if node in node_labels:
                ax.text(
```

```
                *self.node_positions[node, z],
                node_labels[node], *args, **kwargs
            )

    def draw(self):
        self.draw_edges(
            self.edges_within_layers, color="k", alpha=0.3,
            linestyle="-", zorder=2
        )
        self.draw_edges(
            self.edges_between_layers, color="k", alpha=0.3,
            linestyle="--", zorder=2
        )
        for z in range(self.total_layers):
            self.draw_plane(z, alpha=0.2, zorder=1)
            self.draw_nodes(
                [node for node in self.nodes if node[1] == z],
                s=300, zorder=3
            )
        if self.node_labels:
            self.draw_node_labels(
                self.node_labels,
                horizontalalignment="center",
                verticalalignment="center",
                zorder=100,
            )
```

```
paper_nx.nodes()
if __name__ == "__main__":
    node_labels = {
        nn: str(nn) for nn in range(max(len(paper_nx.nodes()),
         len(auth_nx.nodes())))
    }
    fig = plt.figure(figsize=(696, 69))
    ax = fig.add_subplot(111, projection="3d")
    LayeredNetworkGraph(
        [auth_nx, paper_nx, year_nx],
        node_labels=node_labels,
        ax=ax,
        layout=nx.spring_layout,
    )
    ax.set_axis_off()
    plt.show()
```

A.0.3 Attributed Sum

```
import pandas as pd
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import plotly.graph_objects as go
import networkx.algorithms.community as nx_comm
from sklearn.cluster import DBSCAN
from tqdm import tqdm
```

```
get_ipython().system("pip install pandas")
get_ipython().system("pip install -U scikit-learn")
get_ipython().system("pip install plotly")
f1 = open("output1.txt", "r").read()
arr = []
for i in tqdm(f1.split("\n\n")):
    arr.append(i.split("\n"))
arr = arr[:-1]
brr = arr[:]
for i in brr:
    if i[2] == "":
        print(i)
for i in range(len(brr)):
    for j in range(5, 8):
        brr[i][j] = brr[i][j].split()
        brr[i][j] = brr[i][j][0] + ":" + brr[i][j][1]
d = {}
for i in range(len(brr)):
    d[brr[i][0]] = [brr[i][5], brr[i][6], brr[i][7], brr[i][2]]
for i in d:
    d[i][0] = d[i][0].split(":")
    d[i][1] = d[i][1].split(":")
    d[i][2] = d[i][2].split(":")
    k = d[i][-1].split(" ")
    d[i] = {
        d[i][0][0]: float(d[i][0][1]),
```

```
        d[i][1][0]: float(d[i][1][1]),
        d[i][2][0]: float(d[i][2][1]),
        k[0]: " ".join((k[i] for i in range(1, len(k)))),
    }
g = {}
for i in d:
    g[i] = {
        "attr_sum": (5 * d[i]["#hi"] + d[i]["#pi"] + d[i]["#upi"]) // 3,
        "#a": d[i]["#a"],
    }
k = sorted(g.items(), key=lambda x: x[1]["attr_sum"], reverse=True)
x = k[:10000]
at_dic = {}
for i in x:
    at_dic[i[0]] = {"attr_sum": i[1]["attr_sum"]}
attributes = list(your_dict.keys())
values = [item["attr_sum"] for item in your_dict.values()]
threshold = 50
df = pd.DataFrame({"attribute": attributes, "value": values})
df["community"] = (df["value"] - 1) // threshold
dbscan = DBSCAN(eps=0.5, min_samples=5)
clusters = dbscan.fit_predict(df[["value", "community"]])
df["cluster"] = clusters
df.to_csv("your_output_file.csv", index=False)
df = pd.read_csv("your_output_file.csv")
fig = plt.figure()
ax = fig.add_subplot(111, projection="3d")
```

```
colors = ["blue", "red", "green", "yellow",
 "orange", "purple", "pink", "violet"]
for i in range(-1, max(df["cluster"]) + 1):
    cluster_data = df[df["cluster"] == i]
    ax.scatter(
        pd.to_numeric(cluster_data["attribute"].str[3:]),
        pd.to_numeric(cluster_data["value"]),
        cluster_data["community"],
        c=colors[i % len(colors)],
    )
ax.set_xlabel("Attribute")
ax.set_ylabel("Value")
ax.set_zlabel("Community")
plt.show()
df = pd.read_csv("your_output_file.csv")
fig = plt.figure()
ax = fig.add_subplot(111, projection="3d")
colors = ["blue", "red", "green", "yellow",
"orange", "purple", "pink", "violet"]
for i in range(-1, max(df["cluster"]) + 1):
    cluster_data = df[df["cluster"] == i]
    ax.scatter(
        pd.to_numeric(cluster_data["attribute"].str[3:]),
        pd.to_numeric(cluster_data["value"]),
        cluster_data["community"],
        c=colors[i % len(colors)],
    )
```

```
ax.set_xlabel("Attribute")
ax.set_ylabel("Value")
ax.set_zlabel("Community")
plt.show()
fig = go.Figure(
    data=[
        go.Scatter3d(
            x=df["attribute"].str[3:],
            y=df["value"],
            z=df["community"],
            mode="markers",
            marker=dict(
                color=df["cluster"], colorscale="Viridis",
                 opacity=0.7, symbol="circle"
            ),
        )
    ]
)
fig.update_layout(
    scene=dict(xaxis_title="Attribute", yaxis_title="Value",
    zaxis_title="Community")
)
fig.show()
Gf = nx.from_pandas_edgelist(df, source="community", target="value")
print(nx_comm.modularity(Gf, nx_comm.label_propagation_communities(Gf)))
```

A.0.4 Clustering [Similarities]

```
import pandas as pd
import numpy as np
from sklearn.cluster import KMeans
import plotly.graph_objs as go
import plotly.offline as pyo
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics.pairwise import cosine_similarity
import re
import matplotlib.pyplot as plt
import nltk
from tqdm import *
from scipy.cluster.hierarchy import dendrogram, linkage
from mpl_toolkits.mplot3d import Axes3D
import itertools

get_ipython().system("pip install pandas")
get_ipython().system("pip install -U scikit-learn")
get_ipython().system("pip install plotly")
get_ipython().system("pip install modularity")
get_ipython().system("pip install python-louvain")
get_ipython().system("pip install nltk")
f1 = open("output1.txt", "r").read()
arr = []
for i in tqdm(f1.split("\n\n")):
    arr.append(i.split("\n"))
```

```
arr = arr[:-1]
brr = arr[:]
for i in brr:
    if i[2] == "":
        print(i)
brr[0]
for i in range(len(brr)):
    for j in range(5, 8):
        brr[i][j] = brr[i][j].split()
        brr[i][j] = brr[i][j][0] + ":" + brr[i][j][1]
d = {}
for i in range(len(brr)):
    d[brr[i][0]] = [brr[i][5], brr[i][6], brr[i][7], brr[i][2]]
for i in d:
    d[i][0] = d[i][0].split(":")
    d[i][1] = d[i][1].split(":")
    d[i][2] = d[i][2].split(":")
    k = d[i][-1].split(" ")
    d[i] = {
        d[i][0][0]: float(d[i][0][1]),
        d[i][1][0]: float(d[i][1][1]),
        d[i][2][0]: float(d[i][2][1]),
        k[0]: " ".join((k[i] for i in range(1, len(k)))),
    }
g = {}
for i in d:
    g[i] = {
```

```
            "attr_sum": (
                5
                * cmath.rect(math.exp(math.log(abs(d[i]["#hi"]))),
                 cmath.phase(d[i]["#hi"]))
                + cmath.rect(
                    math.exp(math.log(abs(d[i]["#upi"]))),
                     cmath.phase(d[i]["#upi"])
                )
                + cmath.rect(math.exp(math.log(abs(d[i]["#pi"]))),
                 cmath.phase(d[i]["#pi"]))
            ).real
            // 3,
            "#a": d[i]["#a"],
        }
k = list(g.items())
len(k)
affili = []
cs = {}
for author in tqdm(g):
    affiliations_text = g[author]["#a"]
    affiliations_list = re.split(";(?=\\w)", affiliations_text)
    affiliations_list = [
        re.sub("<[^<]+?>|rfc822", "", affiliation).strip()
        for affiliation in affiliations_list
    ]
    affili.append(affiliations_list)
assert len(affili) == len(k)
```

```
po = {}
for ind, i in tqdm(enumerate(affili)):
    try:
        vectorizer = CountVectorizer()
        count_matrix = vectorizer.fit_transform(i)
        cosine_sim = cosine_similarity(count_matrix)
        cs[ind] = sum(cosine_sim)
    except:
        po[ind] = i
po
k[10602]
xyyx = {}
j = 0
for ind, i in tqdm(enumerate(k)):
    if ind in po:
        continue
    else:
        xyyx[j] = i
        j += 1
(xyyx[10602][1], xyyx[10602])
(len(k), len(g), len(xyyx))
ffk = {}
for ind, i in tqdm(enumerate(xyyx)):
    ffk[xyyx[ind][0]] = xyyx[ind][1]
ffk["#index 38264"]
affili = []
cs = {}
```

```
for author in tqdm(ffk):
    affiliations_text = ffk[author]["#a"]
    affiliations_list = re.split(";(?=\\w)",
     affiliations_text)
    affiliations_list = [
        re.sub("<[^<]+?>|rfc822", "", affiliation).strip()
        for affiliation in affiliations_list
    ]
    affili.append(affiliations_list)
poo = {}
for ind, i in tqdm(enumerate(affili)):
    try:
        vectorizer = CountVectorizer()
        count_matrix = vectorizer.fit_transform(i)
        cosine_sim = cosine_similarity(count_matrix)
        cs[ind] = sum(cosine_sim)
    except:
        poo[ind] = i
xyyx[0]
kk = 0
for i, j in zip(xyyx, cs):
    xyyx[i][1]["#a"] = np.round(cs[kk], 1)
    kk += 1
kk = 0
for i, j in zip(ffk, cs):
    ffk[i]["#a"] = np.round(cs[kk], 1)
    kk += 1
```

```
t = dict(
    itertools.islice(
        dict(sorted(ffk.items(), key=lambda x: x[1]["#a"],
         reverse=True)).items(), 10000
    )
)
t["#index 1522914"]
your_dict1 = {}
c1 = 1
for i in t:
    your_dict1[c1] = {"#similarity_score": t[i]["#a"]}
    c1 += 1
print(your_dict1)
your_dict1[14]
len(your_dict1)
cs[0]
affili[0]
dict(itertools.islice(your_dict1.items(), 100))
df
df = pd.DataFrame.from_dict(your_dict1, orient="index")
Z = linkage(df, "ward")
fig = plt.figure(figsize=(10, 8))
ax1 = fig.add_subplot(111)
dendrogram(Z, ax=ax1)
ax1.set_title("Hierarchical Clustering Dendrogram")
ax1.set_xlabel("Data points")
ax1.set_ylabel("Distance")
```

```
fig = go.Figure(
    data=[
        go.Scatter(
            x=[row[0], row[1]], y=[row[2], row[2]],
             mode="lines", name=f"Cluster {i}"
        )
        for (i, row) in enumerate(Z)
    ]
)
fig.update_layout(
    title="Hierarchical Clustering Dendrogram",
    xaxis_title="Data points",
    yaxis_title="Distance",
)
pyo.iplot(fig)
pd.Series(Z[:, 1]).astype(int)
df
Z
df = pd.DataFrame.from_dict(your_dict1, orient="index")
kmeans = KMeans(n_clusters=100, random_state=0).fit(df)
df["cluster"] = kmeans.labels_
fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection="3d")
ax.scatter(
    df.index.values.astype(str),
    df["#similarity_score"],
    df["cluster"],
```

```
    c=kmeans.labels_.astype(float),
)
ax.set_xlabel("Data points")
ax.set_ylabel("Similarity Score")
ax.set_zlabel("Cluster")
plt.show()
df.to_csv("kmeans_clustering_results.csv")
df
df["similarity_score"][1]
type(df.index)
"key1"[3:]
your_dict1
df = pd.DataFrame.from_dict(your_dict1, orient="index")
kmeans = KMeans(n_clusters=3, random_state=0).fit(df)
df["cluster"] = kmeans.labels_
fig = go.Figure(
    data=[
        go.Scatter3d(
            x=df.index.values.astype(str),
            y=df["#similarity_score"],
            z=df["cluster"],
            mode="markers",
            marker=dict(
                size=5,
                color=kmeans.labels_.astype(float),
                colorscale="Viridis",
                opacity=0.8,
```

```
            ),
        )
    ]
)
fig.update_layout(
    title="KMeans Clustering Results",
    scene=dict(
        xaxis_title="Data points", yaxis_title="Similarity Score",
         zaxis_title="Cluster"
    ),
)
pyo.iplot(fig)
df2 = pd.read_csv("kmeans_clustering_results.csv")
kmeans = KMeans(n_clusters=10, random_state=0).fit(df2)
df2["cluster"] = kmeans.labels_
fig = go.Figure(
    data=[
        go.Scatter3d(
            x=df2.index.values.astype(str),
            y=df2["similarity_score"],
            z=df2["cluster"],
            mode="markers",
            marker=dict(
                size=5,
                color=kmeans.labels_.astype(float),
                colorscale="Viridis",
                opacity=0.8,
```

```
            ),
        )
    ]
)
fig.update_layout(
    title="KMeans Clustering Results",
    scene=dict(
        xaxis_title="Data points", yaxis_title="Similarity Score",
         zaxis_title="Cluster"
    ),
)
pyo.iplot(fig)
similarity_scores = []
for key, value in your_dict1.items():
    similarity_scores.append(value["#similarity_score"])
X = np.array(similarity_scores).reshape(-1, 1)
wcss = []
for i in trange(1, len(X)):
    kmeans = KMeans(
        n_clusters=i, init="k-means++", max_iter=30,
        n_init=10, random_state=0
    )
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)
plt.plot(range(1, len(X)), wcss)
plt.title("Elbow Method")
plt.xlabel("Number of clusters")
```

```
plt.ylabel("WCSS")
plt.show()
similarity_scores = []
for key, value in your_dict1.items():
    similarity_scores.append(value["#similarity_score"])
X = np.array(similarity_scores).reshape(-1, 1)
elbow = []
for k in range(1, 10):
    kmeans = KMeans(n_clusters=k).fit(X)
    elbow.append(kmeans.inertia_)
plt.plot(range(1, 10), elbow)
plt.title("Elbow Plot")
plt.xlabel("Number of Clusters")
plt.ylabel("Inertia")
plt.show()
k = 3
kmeans = KMeans(n_clusters=k).fit(X)
labels = kmeans.labels_
for i in range(k):
    print(
        f"Cluster {i}: {[your_dict1[k]['#similarity_score']
        for (k, v) in enumerate(labels) if v == i]}"
    )
```