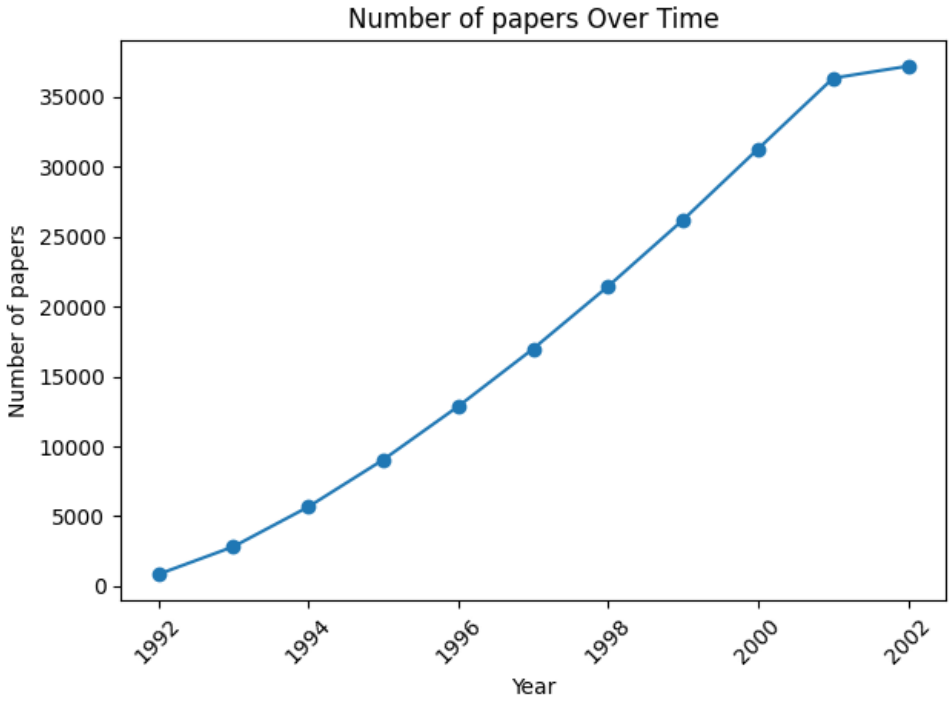
ANALYZING CITATION NETWORKS

Precog Recruitment Task

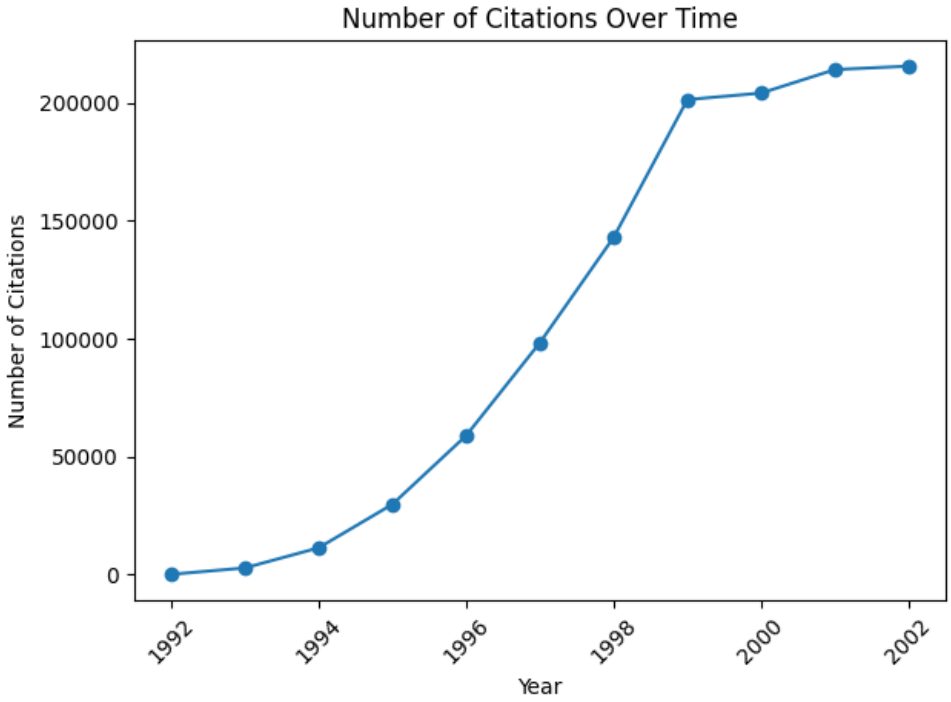
1 Introduction

This project involves exploring the High-energy physics citation network. Arxiv HEP-PH (high energy physics phenomenology) citation graph from the e-print arXiv and covers all the citations within a dataset of 34,546 papers with 421,578 edges. If a paper i cites paper j, the graph contains a directed edge from i to j. If a paper cites, or is cited by, a paper outside the dataset, the graph does not contain any information about this. This dataset is temporal, which means the structure of the network changes over time as new academic papers are published.

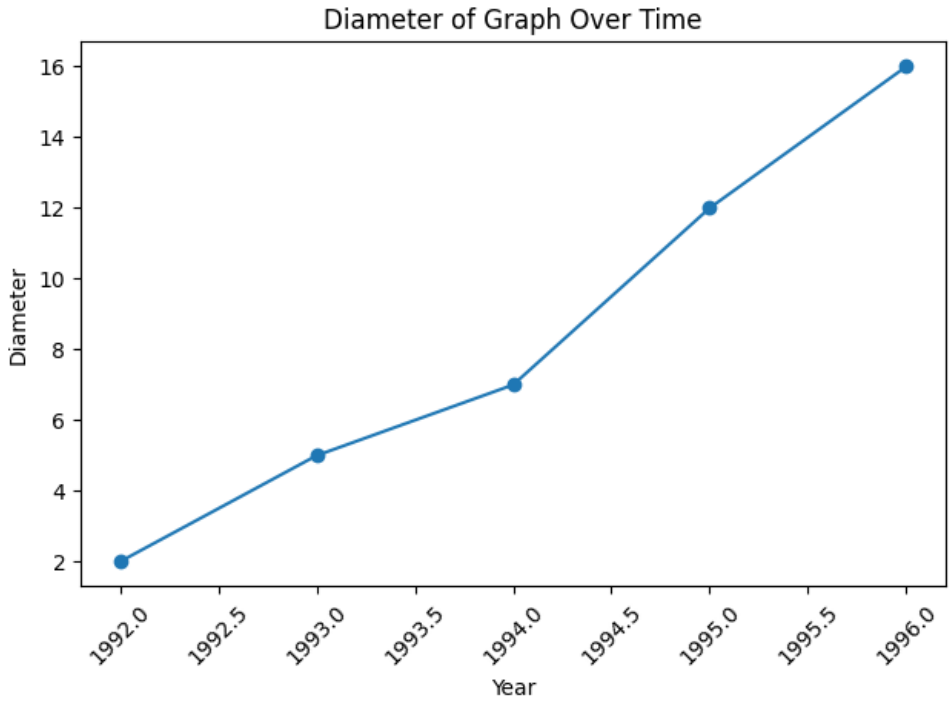
2 Task 1



The above plot just shows how new papers are being added over time. As discussed in Introduction the total number of papers in the given dataset is 34,546 and as you can see in the above plot the number of papers reaches that number by the end of 2001. But it also exceeds the total number of papers, this is because in the given dataset there are some cross-cited papers too.

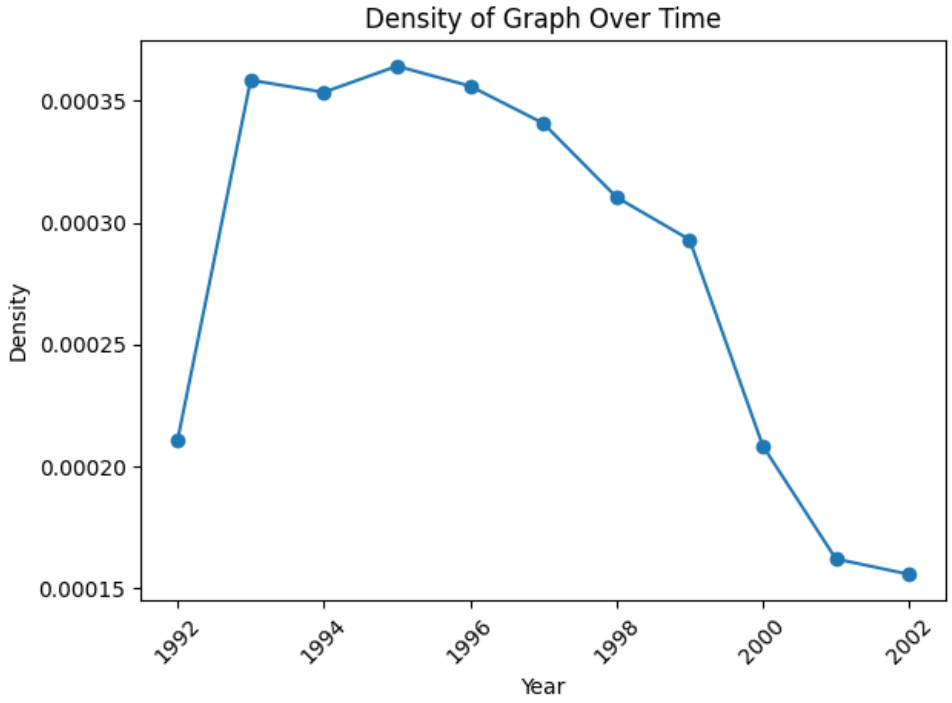


This plot shows how the papers are being cited over time. With time, the number of citations increase. Obviously this number can't decrease so you won't see any dip in this graph. The number of edges in the plot doesn't reach 421,578 because a lot of the edges in the given dataset have fake research id's, i.e, those id's don't have any publication date in the other dataset. So we don't take those nodes/edges into account.

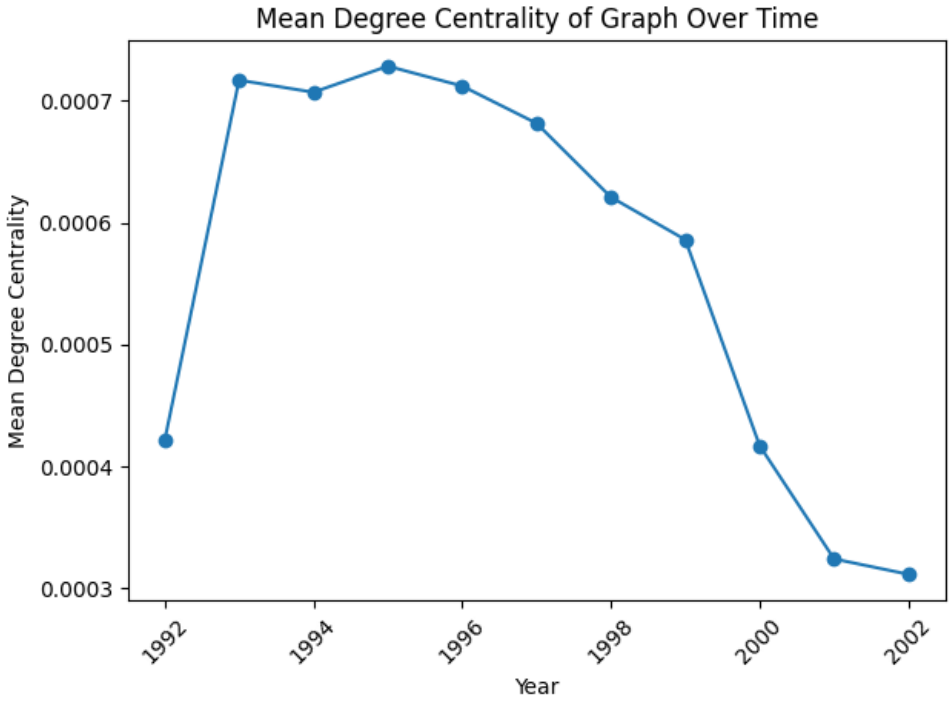


The diameter of the graph is the longest shortest path between any pair of nodes. In other words, it is the maximum number of citations that you would need to follow to get from one paper to another, considering the direction of the citations. Finding the diameter of a directed graph is computationally expensive, as it requires calculating the shortest path between all pairs of nodes. So, I have used 50% of the original dataset to plot the above measure, i.e, $1992 - 1996$.

As you can see the diameter of the graph constantly increases over time going from 2 to 16 , because with time new papers are being published and being cited with the old ones hence increasing the average path length.

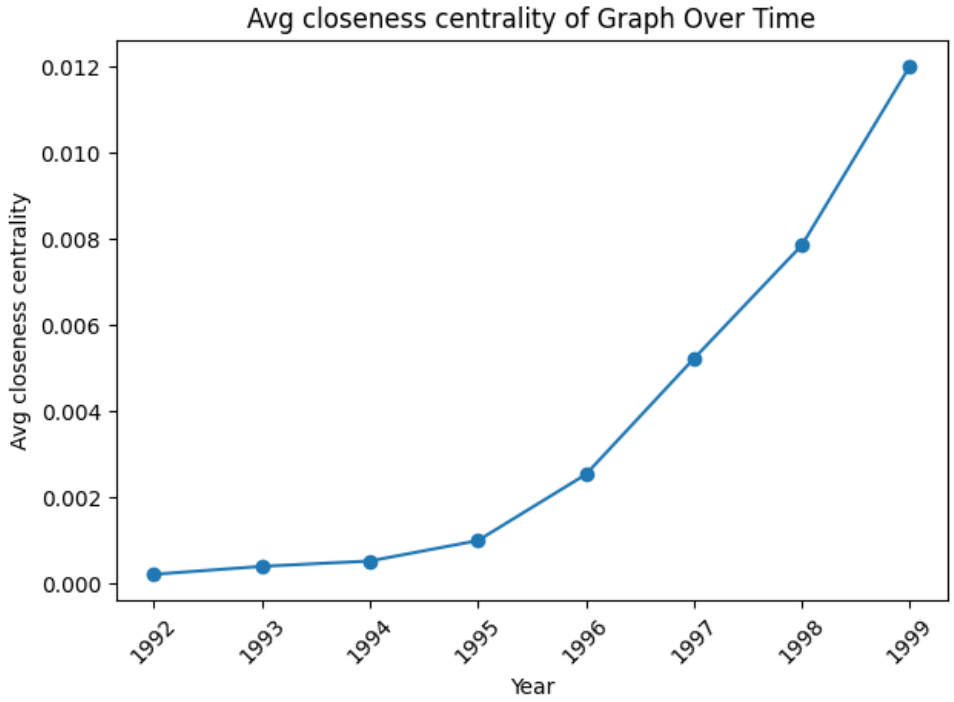


The density of the graph is a measure of how connected the graph is, or how many edges are present relative to the total number of possible edges. The density of a directed graph G with n nodes and m edges can be calculated as: $D = \frac{m}{n(n-1)}$
A high density suggests that many papers cite each other, creating a densely connected network. Conversely, a low density indicates that there are fewer citations relative to the total number of possible citations, resulting in a sparser network. As you can see, it first increases as many papers cited each other in the inital few years, but as time passes by the number of papers increases more rapidly then the number of citations so density decreases.



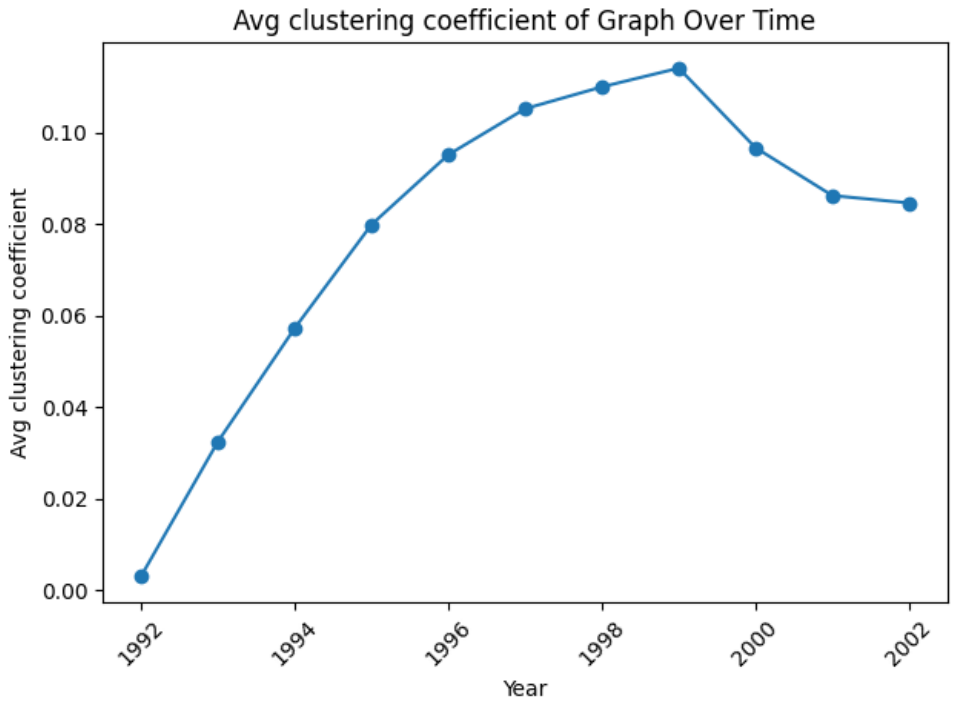
The mean degree centrality is a measure of the average number of citations each paper receives (in-degree) or makes (out-degree). Note that this plot could be made into two: in-degree and out-degree measure to get more insightful information, but I have plotted just the degree combining the both.

The degree centrality of a node v in a directed graph is defined as the number of edges incident to v (in-degree or out-degree), divided by the total number of nodes in the graph minus one (to normalize the value).



The average closeness centrality is a measure of how close each paper is to all other papers in the graph, taking into account the direction of citations. It quantifies the average distance from a node to all other nodes. Closeness centrality of a node is defined as the reciprocal of the sum of the shortest path distances from that node to all other nodes in the graph. Again, as it is computationally expensive operation, I have used 75% of the dataset to plot the graph, i.e, $1992 - 1999$.

This measure provides insight into how well-connected and influential papers are in the citation network. Papers with high average closeness centrality are more central and influential in the network, as they are closer to other papers in terms of citation relationships.



The clustering coefficient of a node in a directed graph measures the extent to which its neighbors are connected to each other. It is a measure of the degree to which nodes tend to cluster together, taking into account the direction of citations.
As you can see, the clustering coefficient increases till the year 1999, but after that a lot of new papers got published which didn't have much citations, so the clustering coefficient was bound to go down.

3 Task 2

3.1 Louvain algorithm

The Louvain algorithm is based on the idea of graph density. Developed by Vincent D. Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre, it's particularly efficient at identifying communities in large networks.

The algorithm works by iteratively optimizing a modularity function that measures the quality of a partition of the network into communities. It starts with each node belonging to its own community and then iteratively merges communities to maximize the modularity. The process continues until no further improvement in modularity can be achieved.

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

- Q is the modularity of the partition.
- A_{ij} is the element in the adjacency matrix of the network.

- k_i is the degree of node i.
- m is the total number of edges in the network.
- c_i is the community to which node i is assigned.
- $\delta(c_i, c_j)$ is 1 if $c_i = c_j$ and 0 otherwise.

  Steps:

  1. Initially, assign each node a unique community such that total nodes = total unique communities
  2. Now, in an iterative manner, we will try assigning every node i to its neighboring node j community and recalculate the modularity of the graph. If modularity improves as compared to when node i wasn't in the j node's community, we will assign i to the community the same as j else not.
  3. This will be repeated for multiple iterations until unless no further gain is observed in modularity by moving any node to its neighbor's community and we have reached a maxima for modularity.
  4. We can then club all the nodes in a single community into one single node and connect with other differnet communities if they had atleast one edge from one of their nodes to one of the destination community nodes.

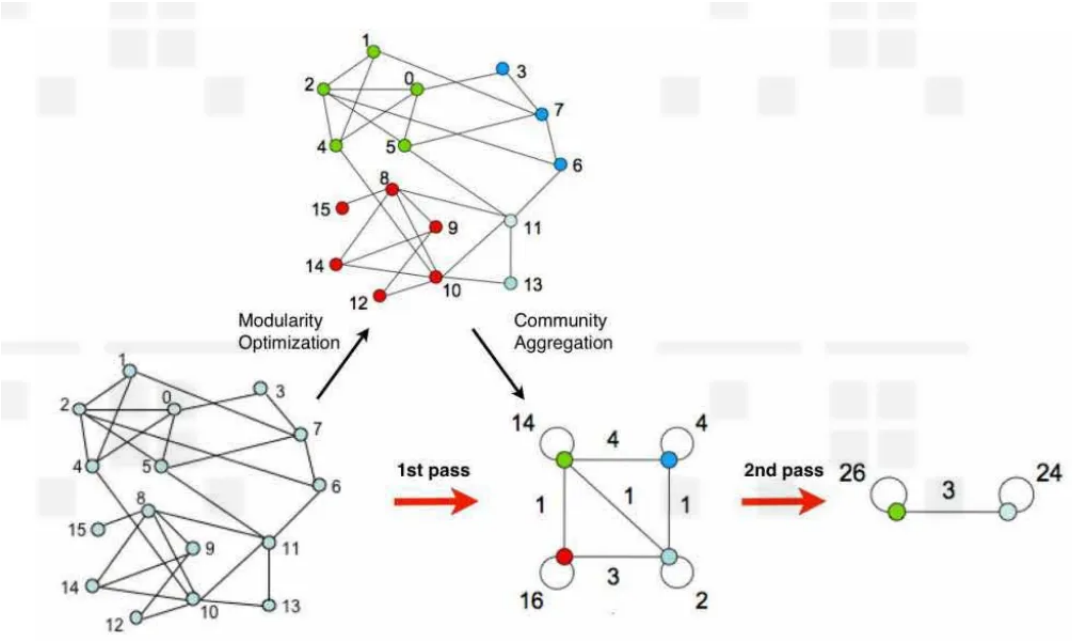
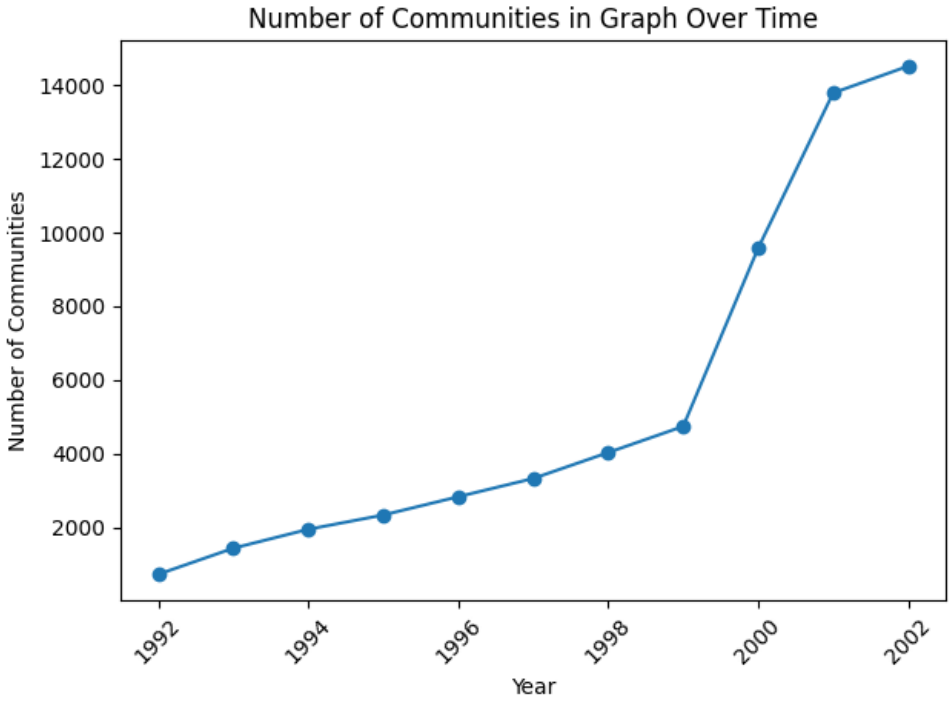


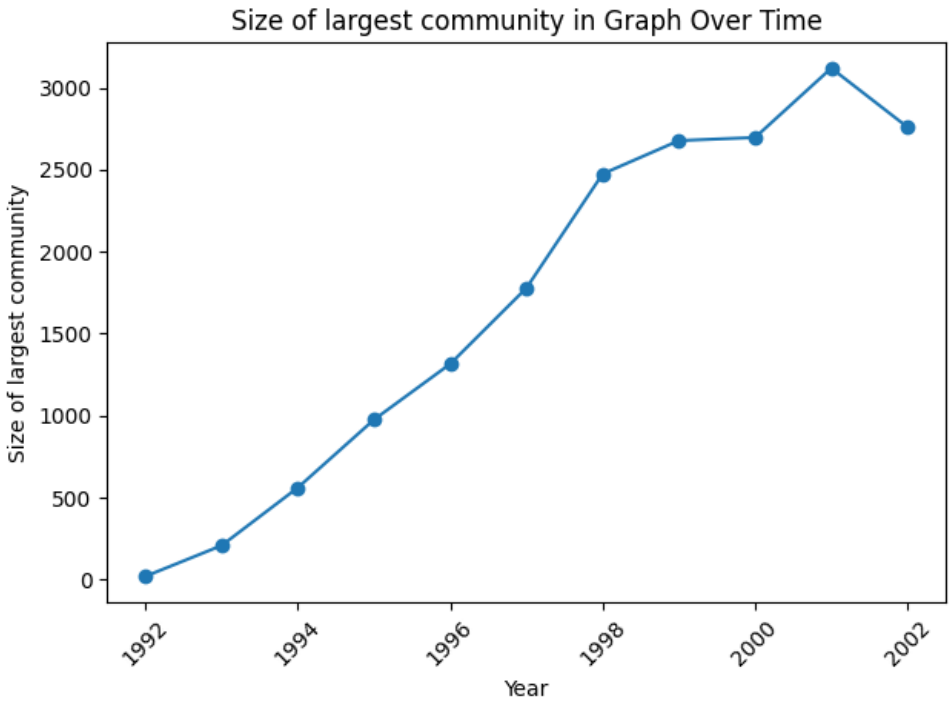
Figure 1: reference

Louvian Algorithm is considered to be quite efficient for large-scale networks. The complexity of each iteration is approximately $O(nlogn)$ where n is the number of nodes in the network.

I used *louvain_communities* function in netowrkx library to get the communities of the graph at a time T. I also plotted number of communities in the graph over time and size of the largest community over time. Apart from that, to analyse the communities, I also drew the different communities at 6 different time periods. Because drawing communities such huge dataset is of no use, I used 1% of the dataset to plot those drawings so that some inferences could be made.



In the above plot, you can see how the number of communities keep increasing as time passes by because there are new papers getting added to the network and hence new communities are being discovered or already made communities are being formed into stronger sub-communities.



In this plot, I have plotted the size of the largest community over time. As you can see, it increases till the year 2001, after which it decreased a bit. This is probably because the largest communities might have broken down into smaller stronger sub-communities.

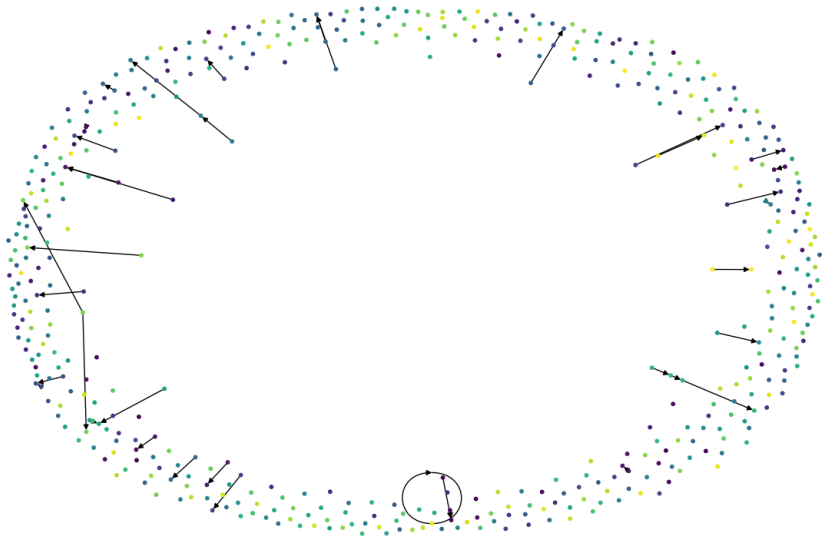


Figure 2: 1992

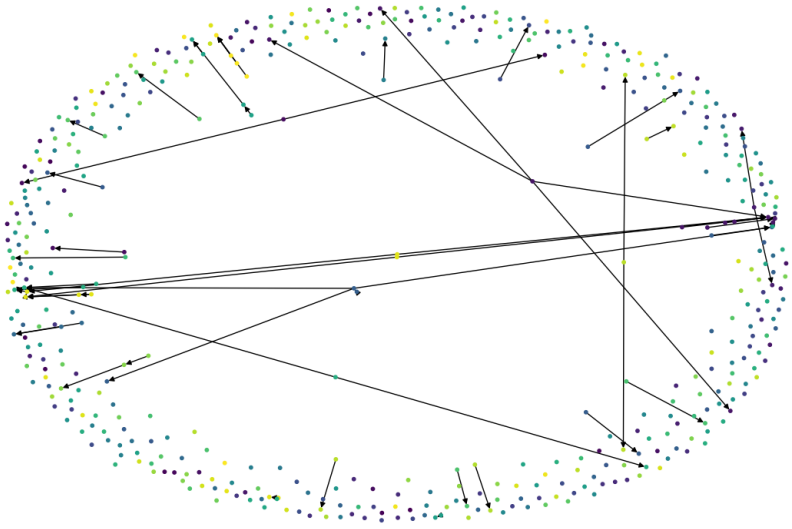


Figure 3: 1994

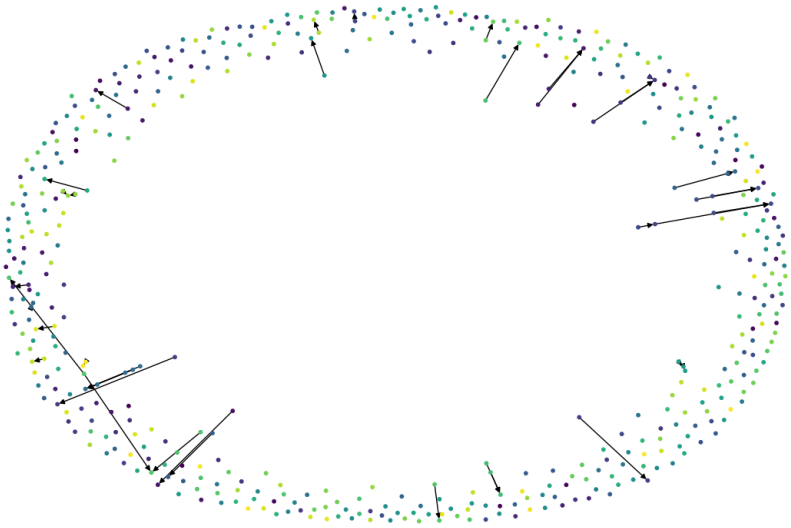


Figure 4: 1996

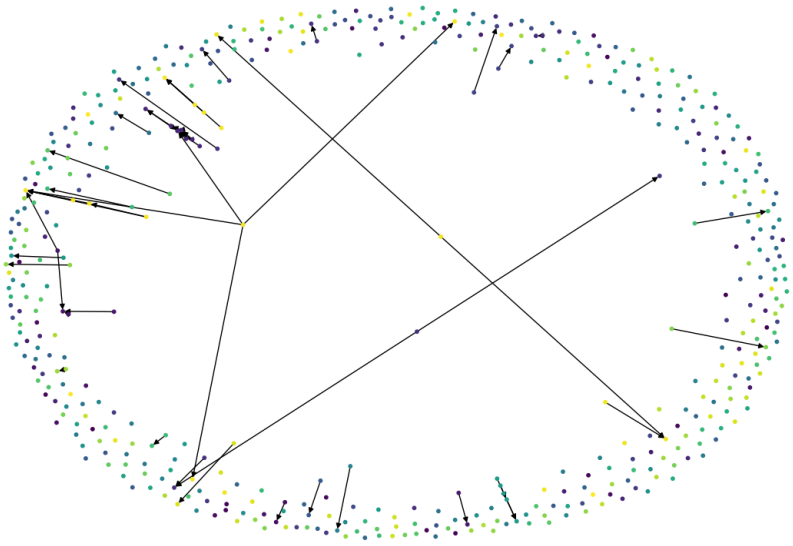


Figure 5: 1998

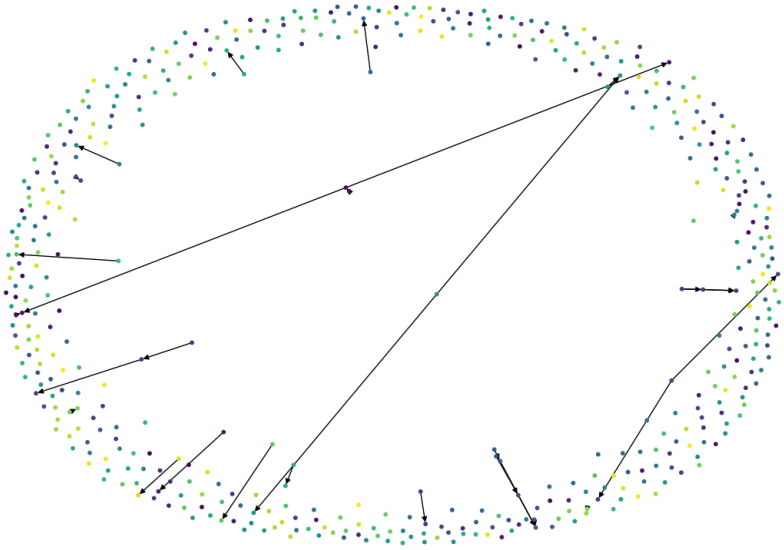


Figure 6: 2000

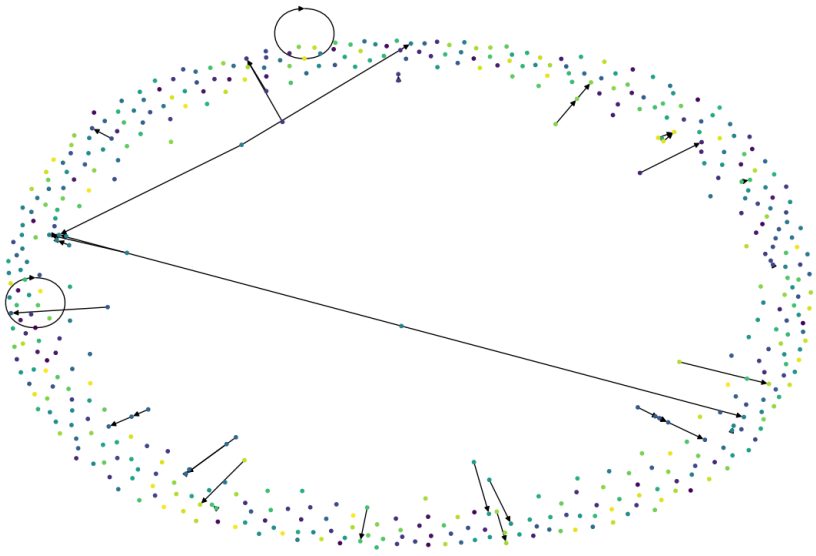


Figure 7: 2002

3.2 Girvan–Newman technique

Girvan-Newman method separates the network based on the betweenness of the edges. Unlike the Louvain algorithm, which optimizes modularity, Girvan-Newman uses a different approach based on edge betweenness centrality.

The algorithm works by iteratively removing the edge with the highest betweenness centrality, recalculating betweenness centrality after each removal. Betweenness centrality of an edge is a measure of the number of shortest paths between pairs of nodes that pass through that edge. By doing this, edges that connect communities (and thus have high betweenness) are identified and removed, gradually breaking the network into smaller components, which represent communities.

Steps:

1. Select a node X, and perform BFS to find number of shortest path from the node X to each node, and assign the numbers as score to each node.
2. Starting from the leaf nodes, we calculate the credit of edge by (1 + (sum of the edge credits to the node))*(score of destination node / score of starting node)
3. Compute the edge credits of all edges in the graph G, and repeat from step 1, until all of the nodes are selected.
4. Sum up all of the edge credit we compute in step 2 and divide by 2, and the result is the edge betweenness of edges.
5. Remove the edges with the highest edge betweenness.

6. Compute the modularity Q (as discussed in previous approach) of the communities split.
7. Repeat from step 1, if Q is greater than 0.3–0.7.

As you can see for each iteration we have to run BFS from each node to find shortest path to every other node. While the Girvan-Newman algorithm is effective in finding communities, its high computational complexity limits its applicability to very large networks. The complexity of the Girvan-Newman algorithm is generally considered to be $O(m^2n)$, where m is number of edges and n is the number of nodes in the network.

I used *girvan_newman* function in netowrkx library to get the communities of the graph at a time T. For this algorithm, I just have used 1% of the dataset. I have plotted number of communities in the graph over time and size of the largest community over time. Apart from that, to analyse the communities, I also drew the different communities at 6 different time periods.

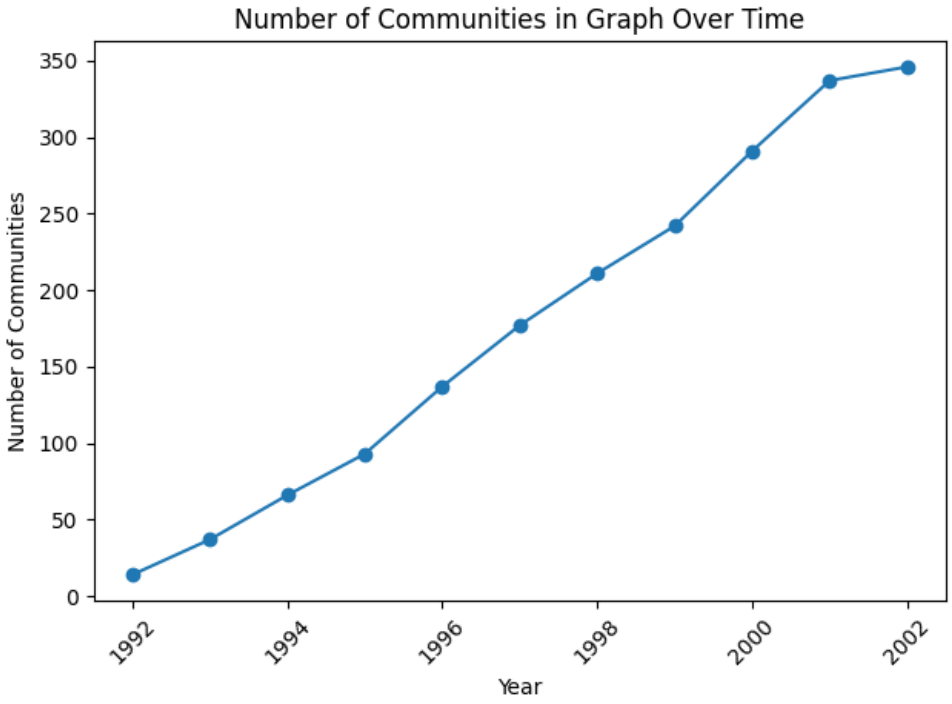


Figure 8: 1% of dataset

Again, the number of communities should increase over time as seen in the above graph.

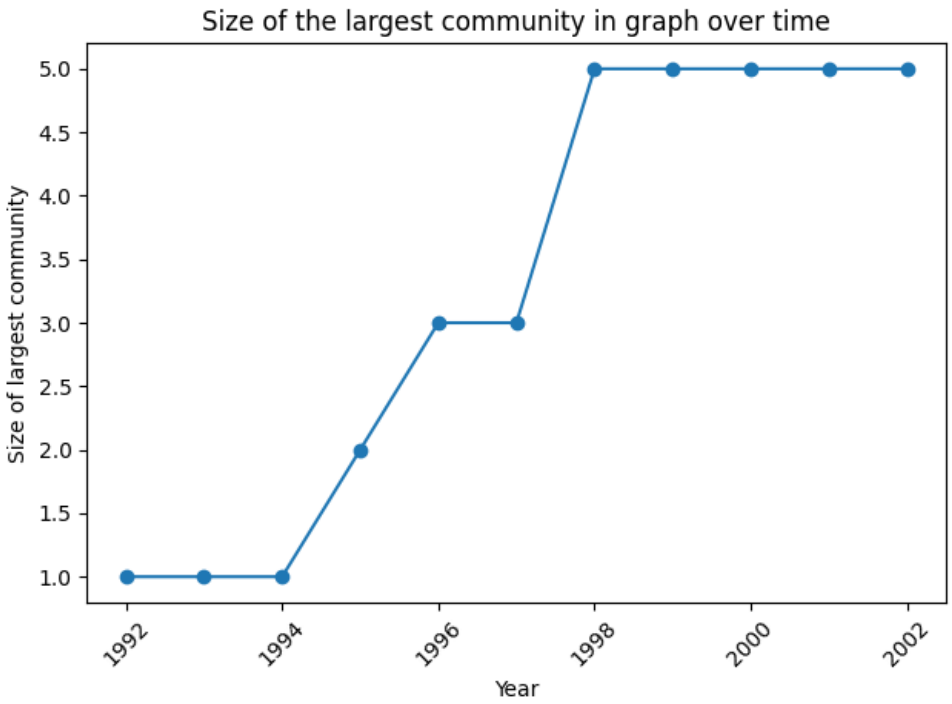


Figure 9: 1% of dataset

The size of the largest community also increases over time while being static for a few years.

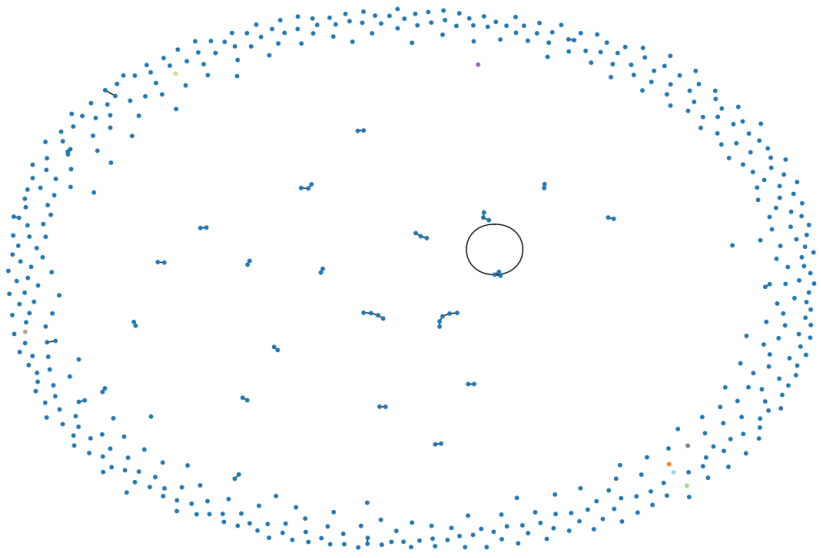


Figure 10: 1992

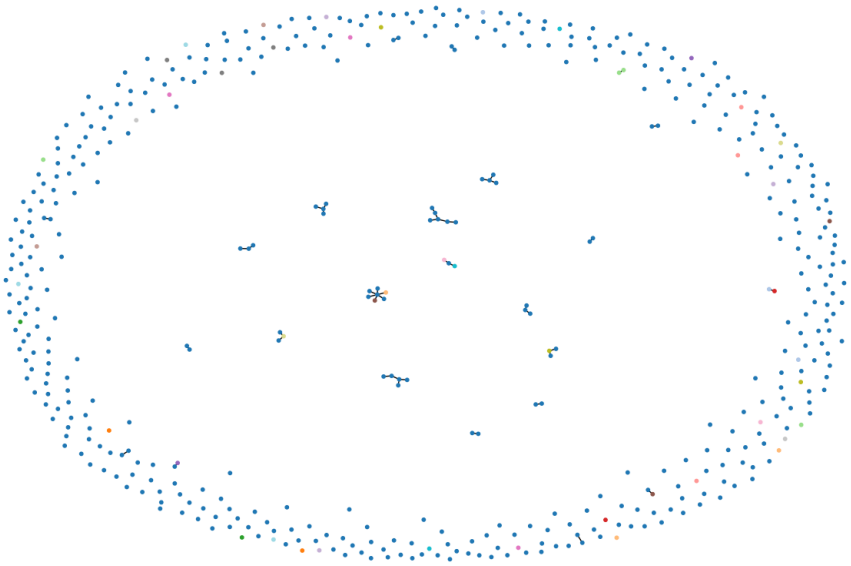


Figure 11: 1994

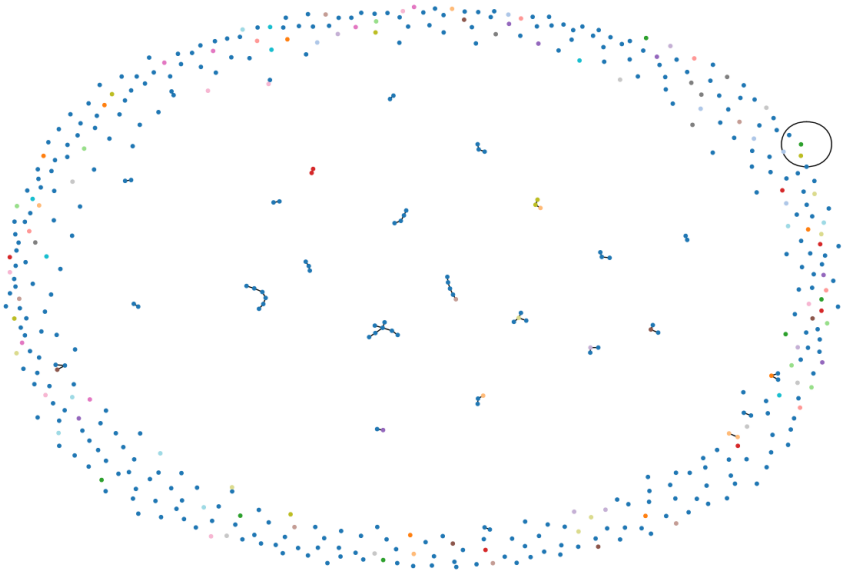


Figure 12: 1996

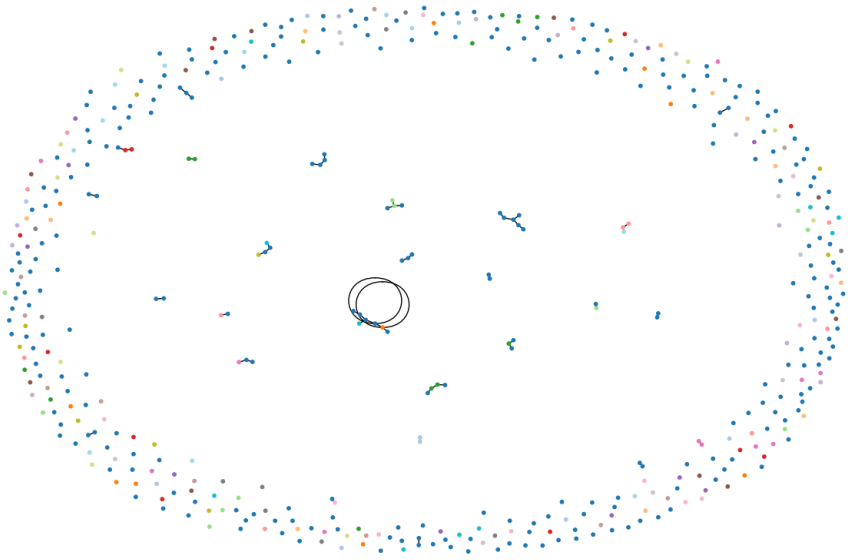


Figure 13: 1998

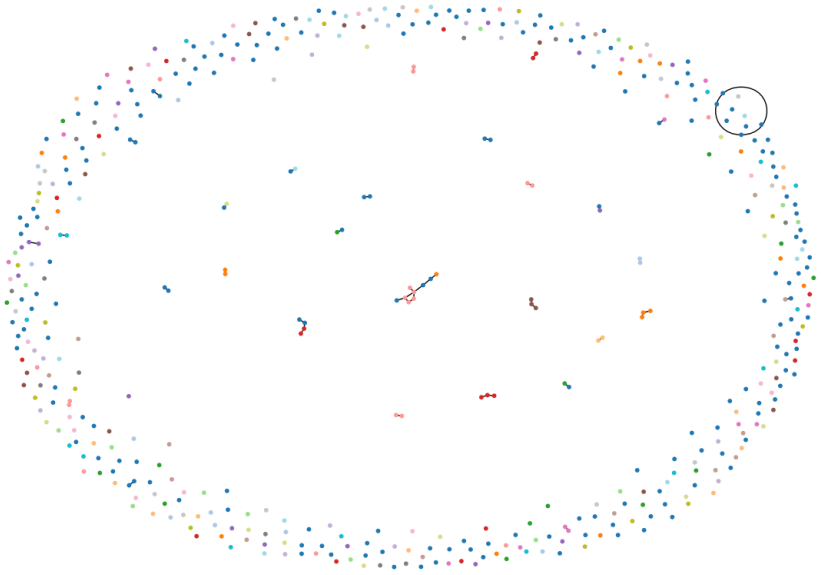


Figure 14: 2000

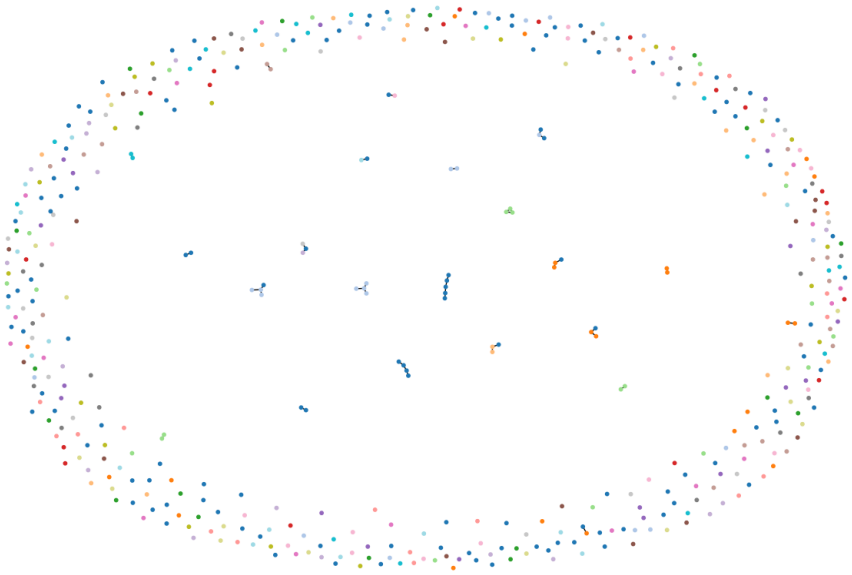


Figure 15: 2002

4 Paper Reading Task

4.1 Summary

From what I could understand, this paper is an extension to the DeepWalk algorithm in the sense that it gives a new way to do random walks in the graph. In deepwalk, the choice of the next node to visit at each step is based on a uniform probability distribution over the neighbors of the current node. But in this paper, they introduce a better way to do random walks so that feature learning can be more appropriate.

So using the method discussed in this paper, we can represent each node as a d-dimensional vector which basically are the features of the node and then we can apply the word2vec model to do downstream prediction tasks. There are two main architectures for training Word2Vec models: Continuous Bag of Words (CBOW) and Skip-gram. Mostly skip-gram model is used as it is more common and better and in this paper also they have mentioned to use skip-gram model after feature learning.

node2vec is an algorithmic framework for learning continuous feature representations for nodes in networks. In node2vec, we learn a mapping of nodes to a low-dimensional space of features that maximizes the likeli-

hood of preserving network neighborhoods of nodes. We define a flexible notion of a node's network neighborhood and design a biased random walk procedure, which efficiently explores diverse neighborhoods.

node2vec is better than deepwalk in the sense that it allows for a flexible algorithm that can learn node representations obeying both principles: ability to learn representations that embed nodes from the same network community closely together, as well as to learn representations where nodes that share similar roles have similar embeddings. This would allow feature learning algorithms to generalize across a wide variety of domains and prediction tasks.

In the paper, they also discuss how feature representations of individual nodes can be extended to pairs of nodes (i.e., edges). In order to generate feature representations of edges, we compose the learned feature representations of the individual nodes using simple binary operators. This compositionality lends node2vec to prediction tasks involving nodes as well as edges.

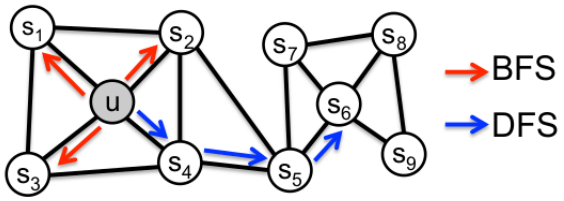
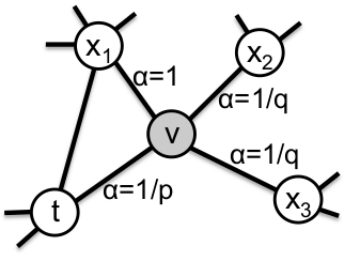


Figure 1: BFS and DFS search strategies from node u ($k = 3$).

In particular, prediction tasks on nodes in networks often shuttle between two kinds of similarities: homophily and structural equivalence. Under the homophily hypothesis, nodes that are highly interconnected and belong to similar network clusters or communities should be embedded closely together (e.g., nodes s1 and u in belong to the same network community). In contrast, under the structural equivalence hypothesis nodes that have similar structural roles in networks should be embedded closely together (e.g., nodes u and s6 act as hubs of their corresponding communities).

There are two extreme sampling strategies for generating neighborhood set(s) N_S of k nodes: BFS and DFS. And the framework which this paper

discusses allows us to do both by fixing the hyperparameters.



To do a 2^{nd} order random walk, fix two parameters p and q which guide the walk: Consider a random walk that just traversed edge (t, v) and now resides at node v. The walk now needs to decide on the next step so it evaluates the transition probabilities π_{vx} on edges (v, x) leading from v. We set the transition probability to $\pi_{vx} = \alpha_{pq}$ (t, x) $\cdot\ w_{vx}$, where

$$\alpha_{pq}(t,x) = \begin{cases} \frac{1}{p} & \text{if } d_{tx} = 0 \\ 1 & \text{if } d_{tx} = 1 \\ \frac{1}{q} & \text{if } d_{tx} = 2 \end{cases}$$

d_{tx} denotes the shortest path distance between nodes t and x.

Intuitively, parameters p and q control how fast the walk explores and leaves the neighborhood of starting node u. In particular, the parameters allow our search procedure to (approximately) interpolate between BFS and DFS and thereby reflect an affinity for different notions of node equivalences.

p is known as the return parameter. It controls the likelihood of immediately revisitng a node in the walk. Setting it to a high value ensures that we are less likely to sample an already visted node in our walk.

q is known as the in-out parameter. It allows to differentiate between to as you want to do dfs or bfs in a vague sense. If $q > 1$, the random walk is biased towards the node closer to node t (hence towards homophily). If $q < 1$, the walk is more inclined to visit nodes which are farther away

from node t (hence towards structural equivalence).

In the paper, they have then discussed the advantages of random walk and given some binary operations between two nodes so that we can get features for an edge too. In the end, they have discussed some applications of node2vec like Perturbation Analysis and Link prediciton.

4.2 Three major strengths of the paper:

- Efficiency: The algorithm is computationally efficient, leveraging a biased random walk strategy to efficiently generate node embeddings/features. This allows node2vec to scale to large networks and learn high-quality embeddings even for massive graphs. This is faster than DeepWalk but there are GNN's which are still faster.

- Flexibility: node2vec introduces a flexible way to explore the neighborhood of a node by balancing between breadth-first search (BFS) and depth-first search (DFS). This allows the algorithm to capture both local and global network structures, leading to more informative node embeddings.

- Flexibility in parameter tuning: While sensitivity to hyperparameters can be seen as a weakness, it also provides flexibility. Researchers and practitioners can fine-tune these parameters based on the specific characteristics of the network and the task at hand, potentially improving performance.

4.3 Three major weaknesses of the paper:

- Fixed Walk Length: Node2Vec relies on fixed-length random walks. Longer walks may miss local structures, while shorter walks may not capture global context effectively.

- Hyperparameters: Tuning parameters like walk length, context size, and dimensions can be challenging. Suboptimal choices may lead to subpar embeddings.

-Order Independence: Unlike text, graph nodes lack a natural order. Node2Vec assumes that the order of neighbors in a walk doesn't matter, which may not always hold true.

4.4 Three improvements to the paper, that would improve the paper:

- Adaptive Walk Length: Instead of fixed-length walks, consider adaptive walk lengths based on node characteristics or graph properties.

- Graph Convolutional Networks (GCNs): Explore combining Node2Vec with GCNs for more expressive embeddings and faster runtimes.

- The paper only talks about node/edge level embeddings. We can introduce graph level embeddings too to make new applications.