A COMPARITIVE STUDY ON NETWORK ANALYSIS

AASTHA PATEL & SHRESHTHA MANKALA

ABTRACT

The project aims at comprehensively analyze network properties such as degree distribution, largest connected component size, clustering coefficient, and shortest path length. To achieve this goal, the analysis with many networks, including categories such as Human social Network, Animal Network, Road Network, Authorship Network, Citation/Collaboration Network, and Interaction Network, Infrastructure Network, etc. by using NetworkX and Snap.py. By examining different types of networks with various styles such as undirected, directed, bipartite and unipartite, weighted, and unweighted graphs, we will focus on grouping the graphs based on their common characteristics and made comparison amongst the result of both the packages.

INTRODUCTION

Network analysis is a valuable method for gaining insights into the structure and behavior of complex systems. The comprehensive analysis of the properties for different types of networks, including Interaction Network, Animal Network, Authorship Network, Affiliation Network, Human Contact Network, Computer Network, Hyperlink Network, Infrastructure Network, Metabolic Network, Miscellaneous Network, Online Social Network and Software Network. The most important properties of networks, including Degree Distribution, Connected Components, Clustering Coefficient, and Shortest Path Length, is examined. The insights gained through network analysis can be applied to a wide range of fields such as finance, engineering, biology, or medical sciences and many more. This comparative study is interesting because it offers a way to gain a deeper understanding of complex systems and their properties, which lead to numerous applications across various fields. The focus is also on comparing the results obtained from NetworkX and Snap.py packages.

RELATED WORK

The paper SNAP: A General-Purpose Network Analysis and Graph Mining Library[1], focuses mainly on the SNAP system and the comparison between the different systems such as Pajek, NetworkX and iGraph. SNAP is built for faster processing and uses less memory, which makes it better than any other existing systems. SNAP provides efficient implementations of traditional as well as the most recent algorithms that make use of machine learning for graph and network analysis. It includes graph creation, manipulation, and analytics methods such as community detection, statistical modeling of networks, network link and missing node prediction, and network structure inference. Overall, through this paper, we learnt about advantages of using Snap.py and how different metrics such as memory consumption and time requirements can be measured.

METHODOLOGY

1. To analyze and visualize the graphs and networks, two Python libraries - NetworkX and Snap.py are used. These libraries are powerful and support many graph models and network properties. The difference between both the libraries is that NetworkX is more flexible when it comes to creation and manipulation of the graph structure which makes it slower. Whereas Snap.py is built to handle large and complex networks better, which can contain up to billions of nodes, which makes it slower for manipulation but is great for faster and efficient computation.

2. The data was extracted from KONECT Network Collection, and all the files are in the 'out.' format, which we chose because of the smaller file size as compared to other formats. The networks chosen are as small as containing 40 nodes and ranging up to as large as 62000 nodes.

3. We analyzed the following network properties: degree distribution, clustering coefficient, connected components, and shortest path length using the following methods.

   - Degree Distribution : It represents the likelihood that a node in the network will have a specific degree and it helps to understand the distribution of connections among nodes.
   - Clustering Coefficient : It measures the extent to which the neighbors of a node are also connected to each other.
   - Connected Components: It identifies groups of nodes that are highly connected to each other but not to other parts of the network.
   - Shortest Path Length: It refers to the smallest number of edges that need to be traversed to travel from one vertex to the other. If there is no path between the vertices, the distance is infinite.

4. To begin with, we imported the necessary libraries.

```
In [1]: import matplotlib.pyplot as plt
        from matplotlib.ticker import MaxNLocator
        import time
        import sys
        import numpy as np
        plt.style.use('ggplot')
        #plt.style.use('seaborn-ticks')
        #plt.style.use('seaborn-notebook')
        plt.rcParams['lines.linewidth']=3
        plt.rcParams['xtick.labelsize']=12
        plt.rcParams['ytick.labelsize']=12
        plt.rcParams['axes.labelsize']=14
```

5. **Loading the dataset**

```
file=dataset
cite = nx.read_adjlist (file)
```

NetworkX

```
G = snap.LoadEdgeList(snap.PUNGraph, dataset)
```

Snap.py

6. Implemented on 20 real life networks. The table below shows the different inbuilt methods used for NetworkX and Snap.py used to calculate the network properties.

Properties	NetworkX	Snap.py
Degree Distribution	Graph.degree()	Graph.GetDeg()
Connected Components	nx.connected_components(Graph)	Snap.GetSccs(Graph, components)
Clustering Coefficient	nx.clustering(Graph)	snap.GetNodeClustCf(Graph, Node.GetId())
Shortest Path Length	nx.all_pairs_shortest_path_length(Graph)	snap.GetShortPath(Graph, node_id, snap.TIntH())

7. Using time and sys packages the time and memory usage for all real time networks was calculated, and the results were tabulated.

```
#Degree_Distribution
cite_time=[]
start = time.time()
Degree_Distribution(cite)
print("Average Degree:",avg_degree(cite))
end=time.time()
s_time=np.round(end-start,5)
cite_time.append(s_time)


#CC_Distribution
start = time.time()
CC_Distribution(cite)
end=time.time()
s_time=np.round(end-start,5)
cite_time.append(s_time)


#Clustering_Analysis
start = time.time()
Clustering_Analysis(cite)
end=time.time()
s_time=np.round(end-start,5)
cite_time.append(s_time)


#Shortest_Path_Analysis
import time
start = time.time()
ShortestPaths_Analysis(cite,5)
end=time.time()
s_time=np.round(end-start,5)
cite_time.append(s_time)
```

8. **Plotting:**
   - Scatter Plot was plotted using Matplotlib to represent the network properties.
   - Line chart is plotted for memory usage based on different package and number of edges.

EXPERIMENTAL SETUP

We conducted our network analysis using the Python programming language (version : Python 3.8.10), leveraging the built-in libraries and packages for exploratory analysis on networks, such as NetworkX and Snap.py. We used Jupyter Notebook as our primary tool for data analysis and visualization. We did not implement any machine learning algorithms for our analysis; therefore, we did not have any hyperparameters to consider. Instead, we focused on analyzing the structure and behavior of the networks, considering only their edges and nodes. We used a variety of networks in our analysis, including interaction networks, animal networks, affiliation networks and so on. We sourced the data for these networks from various publicly available datasets, ensuring that we had a diverse range of properties to analyze. Our analysis was carried out on standard laptop with an Intel Core i7 processor, and 16GB of RAM.

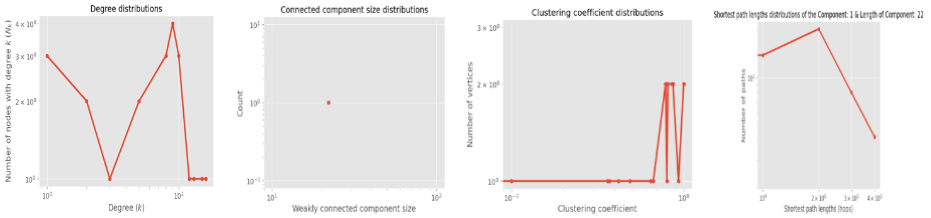
The table below lists the different types of data used in our analysis and their respective properties. All data files are in the "out." format and were obtained from the KONECT Networks dataset.

Category	Name	Node Meaning	Edge Meaning	Network Format	Edge Type
Interaction	Southern Women	Woman, event	Participation	Bipartite, undirected	Unweighted, no multiple edged
	American Football	Team	Game	Unipartite, Undirected	Unweighted, no multiple edged
Affiliation	Corporate Club Membership	Person, organization	Membership	Bipartite, undirected	Unweighted, no multiple edged
Animal	Kangaroos	Kangaroo	Interaction	Unipartite, undirected	Positive weights, no multiple edge
	Dolphins	Dolphin	Association	Unipartite, Undirected	Unweighted, no multiple edges
Authorship	Wiktionary edits	User, Article	Edit	Bipartite, Undirected	Unweighted, multiple edges
Human Contact	Windsurfers	Person	Contact	Unipartite, Undirected	Positive weights, no multiple edge
	HIV	AIDS patient	Sexual contact	Unipartite, Undirected	Unweighted, no multiple edges
Hyperlink	Google.com Interval	Page	Hyperlink	Unipartite, Directed	Unweighted, no multiple edges
Infrastructure	Chicago	Node	Road	Unipartite, Directed	Unweighted, no multiple edges
	Euroroads	City	Road	Unipartite, Undirected	Unweighted, no multiple edges
Computer	Gnutella (31)	Host	Connection	Unipartite, Directed	Unweighted, no multiple edges
Metabolic	Human Proteins (Vidal)	Protein	Interaction	Unipartite, Undirected	Unweighted, no multiple edges
	Yeast	Protein	Interaction	Unipartite, Undirected	Unweighted, no multiple edges
Miscellaneous	Similarities	Concept	Similarity	Unipartite, Undirected	Unweighted, no multiple edges
	Marvel	Character, work	Appearance	Bipartite, Undirected	Unweighted, no multiple edges
	Gene Fusion	Gene	Observed fusion during oncogenesis	Unipartite, Undirected	Unweighted, no multiple edges
Online Social	Film Trust	User	Trust	Unipartite, Directed	Unweighted, no multiple edges
Software	JUNG/Javax	Class	Dependency	Unipartite, Directed	Unweighted, multiple edges

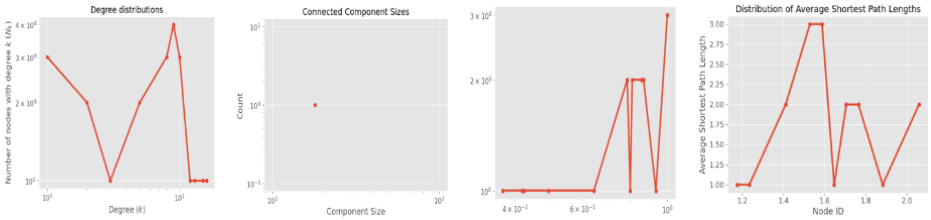
RESULTS

1. Interaction Network : Southern Women (No. of Nodes: 22, No. of Edges: 83)

   NetworkX Package Results:

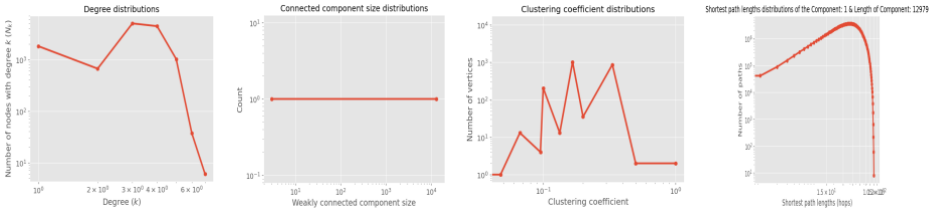


   Snapy.Py Package Results:

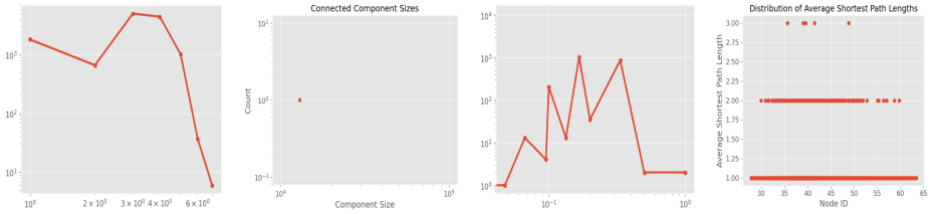


2. Infrastructure Network : Chicago Regional(No. of Nodes: 12982, No. of Edges: 20629)

   NetworkX Package Results:

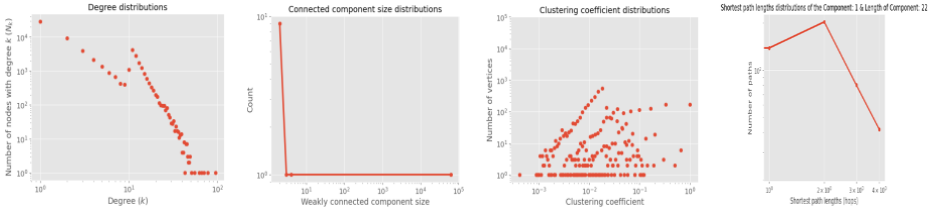


   Snapy.Py Package Results:

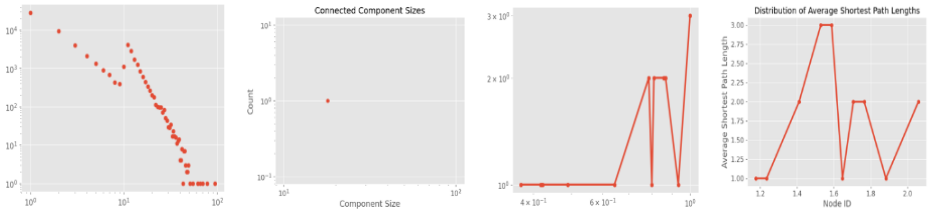


3. Computer Network : Gnutella31 (No. of Nodes: 62590, No. of Edges: 147896)

   NetworkX Package Results:

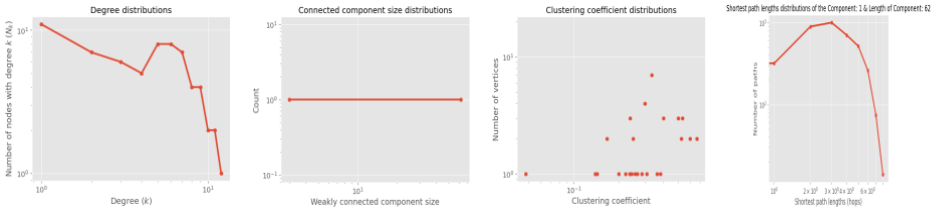


Snapy.Py Package Results:

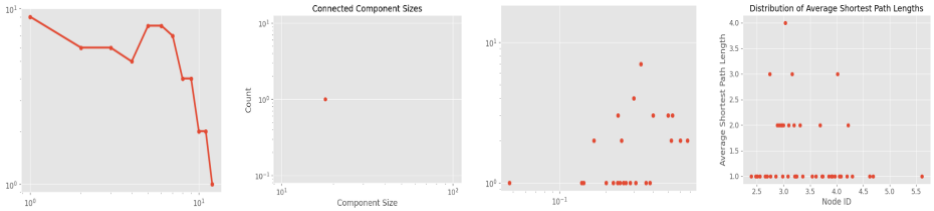


4. Animal Network : Dolphins (No. of Nodes: 65, No. of Edges: 161)

NetworkX Package Results:

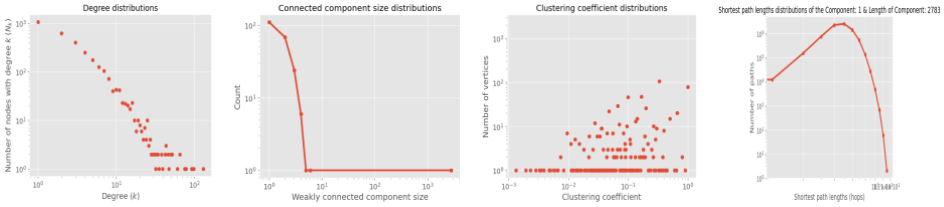


Snapy.Py Package Results:

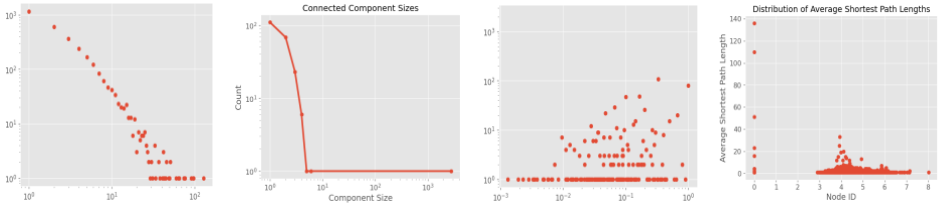


5. Metabolic Network : Human Proteins (No. of Nodes: 3136, No. of Edges: 6728)

NetworkX Package Results:

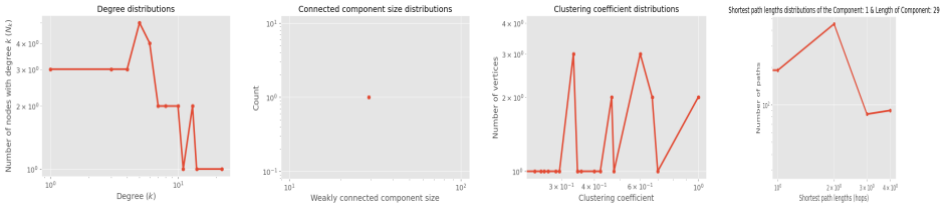


Snapy.Py Package Results:

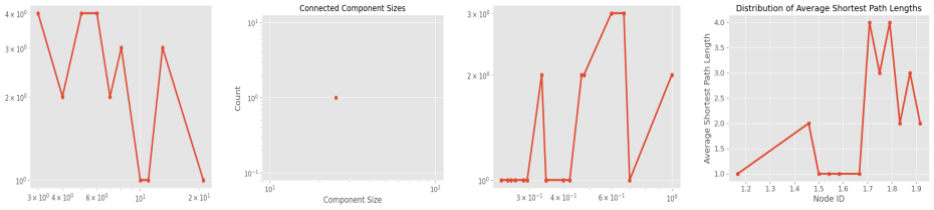


6. Affiliation Network : Club Membership (No. of Nodes: 29, No. of Edges: 98)

NetworkX Package Results:

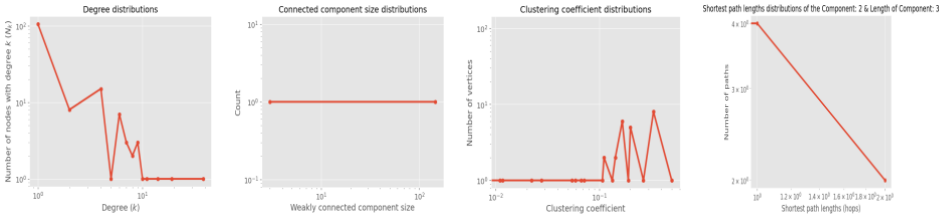


Snapy.Py Package Results:

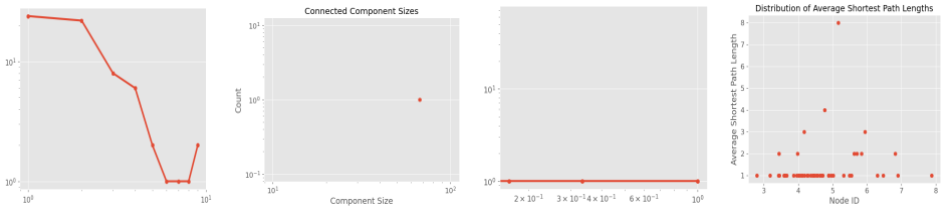


7. Authorship Network : Wiktionary Edits (No. of Nodes: 149 , No. of Edges: 192)

NetworkX Package Results:

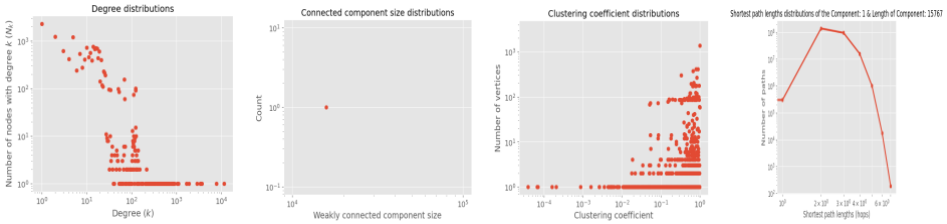


Snapy.Py Package Results:

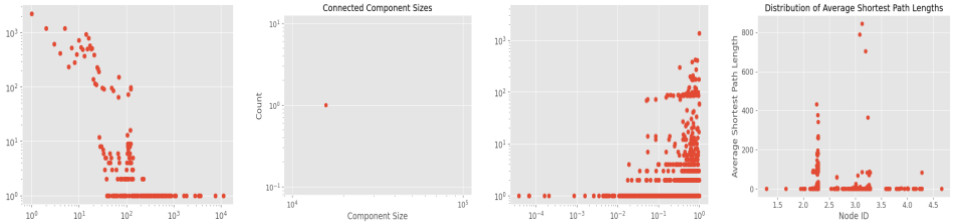


8. Hyperlink Network : Google (No. of Nodes: 15767, No. of Edges: 149461)

NetworkX Package Results:

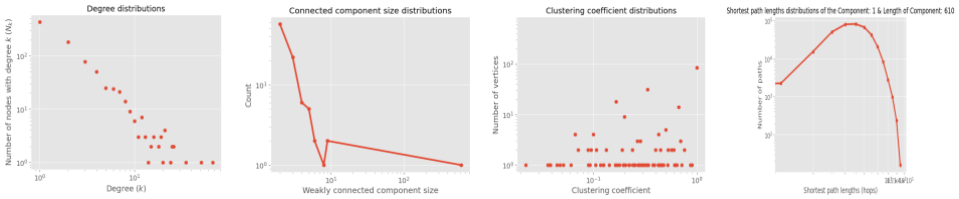


Snapy.Py Package Results:

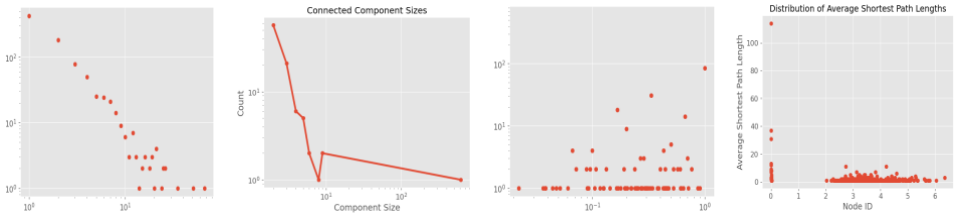


9. Online Social Network : Film Trust (No. of Nodes: 877, No. of Edges: 1311)

NetworkX Package Results:

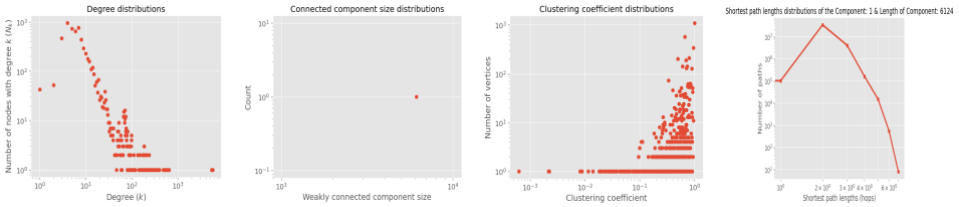


Snapy.Py Package Results:

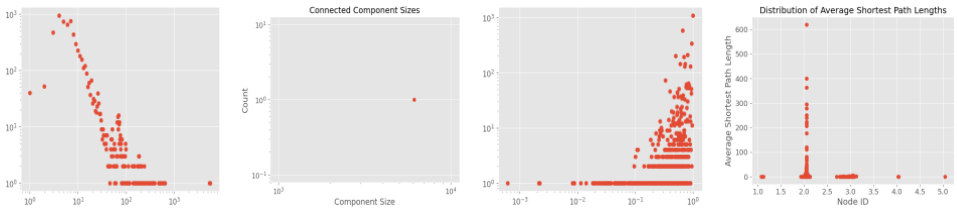


10. Software Network : JUNG/Javax (No. of Nodes: 6124, No. of Edges: 50294)

NetworkX Package Results:



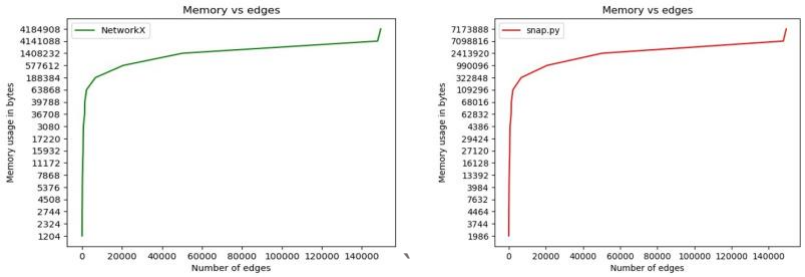
Snapy.Py Package Results:



Due to page limitations, we were only able to display 10 results for each different network type out of the total 20 results. However, we observed that Snap.py performed better with larger nodes. Additionally, it took less time

to execute on larger networks compared to Networkx. We also included a memory usage graph for both the packages. It's worth noting that although Networkx used less memory, the results might be different if we had used a distributed system since Snap.**py operated on distributed systems** while we were using a centralized system.

Memory Usage Graph:



CONCLUSION

Key Findings:

1. The Degree Distribution, Clustering Coefficient and Connected Components have the same results for both the libraries, but there was a slight variation in shortest path length distribution.
2. Memory Usage is more for storing the edges in case of Snap.py in comparison to NetworkX.
3. Snap.py should have performed better in memory consumption and even faster than it already did, because snap.py is design for distributed computing, but in this study, we used centralized systems.
4. Snap.py works efficiently to calculate all properties except for Clustering Coefficient, as it takes comparatively more time than NetworkX.
5. Network under the same categories does not mean they have similar properties hence it does not have any significance.
6. Snap.py works best for large graphs, that is it calculates the distribution for all the properties faster for larger graphs, which have thousands and millions of nodes and edges.

Limitations:

The Snap.py was not able to handle large text files as an input and hence alternative file formats were chosen for minimal crashes. Shortest path length calculations for Snap.py and NetworkX both took too long to compute and for Snap.py we were not able to find out shortest path length for different components.

Future Work:

As Snap.py has been designed for large and distributed computing, the results may vary and show significant improvement if carried out on a distributed system. The study on how a directed and undirected edges, as well as weighted and unweighted edges should be taken into consideration while analyzing the network properties.

TEAM MEMBER CONTRIBUTION:

1.	Dataset Search	Aastha Patel
2.	NetworkX Properties	Aastha Patel
3.	Snap.py Properties	Shreshtha Mankala
4.	Plotting graphs and tabulated results	Shreshtha Mankala, Aastha Patel

REFERENCES:

[1] Leskovec J, Sosič R. SNAP: A General Purpose Network Analysis and Graph Mining Library. ACM Trans Intell Syst Technol. 2016 Oct; Epub 2016 Oct 3. PMID: 28344853; PMCID: PMC5361061.

[2] Barabasi AL, Albert R. Emergence of scaling in random networks. Science. 1999 Oct 15;286(5439):509-12. PMID: 10521342.

[3] Gonzalez JE, Low Y, Gu H, Bickson D, Guestrin C. PowerGraph: Distributed graph-parallel computation on natural graphs. USENIX Symposium on Operating Systems Design and Implementation (OSDI) 2012;12:2.

[4] M. Hay, C. Li, G. Miklau and D. Jensen, "Accurate Estimation of the Degree Distribution of Private Networks," 2009 Ninth IEEE International Conference on Data Mining, Miami Beach, FL, USA, 2009, pp. 169-178.

[5] B. Tao, H. -N. Dai, J. Wu, I. W. -H. Ho, Z. Zheng and C. F. Cheang, "Complex Network Analysis of the Bitcoin Transaction Network," in IEEE Transactions on Circuits and Systems II: Express Briefs, vol. 69, no. 3, pp. 1009-1013(March 2022).